Statistics for Linguists: A patient, slow-paced introduction to statistics and to the programming language R

Schneider, Gerold ; Lauber, Max

Abstract: An interactive E-book with 50 movies and extensive exercises.

STATISTICS FOR LINGUISTS

STATISTICS FOR LINGUISTS

A PATIENT, SLOW-PACED INTRODUCTION TO STATISTICS AND TO THE PROGRAMMING LANGUAGE R

GEROLD SCHNEIDER AND MAX LAUBER

Zürich

Contents

Introduction: Why Statistics for Linguists? 1

Part I. The Foundations

1. Importing and Retrieving Corpus Data: First Steps in R 5
2. Introduction to R 11
3. Basics of Descriptive Statistics 21

Part II. Inferential Statistics

4. Introduction to Inferential Statistics 33
5. The T-Test 43
6. The Chi-Square Test 55

Part III. Language Models

7. 1-Grams 65
8. N-Grams 73

Part IV. Advanced Methods

9. Linear Regression 81
10. Logistic Regression 95
11. Document Classification 103
12. Topic Modelling 107

Part V. The Next Step

13. Search Queries 115

Introduction: Why Statistics for Linguists?

The saying that one cannot study anything today without studying statistics definitely holds true in linguistics. Some of you may roll your eyes, and close the book, but we would advise against doing so. Not only because it would be shame if we lost your attention at the word go, but because learning statistics means learning methodologies which enable you to produce better research, in linguistics and beyond.

Maybe it helps to consider an example. You might notice a conspicuous pattern in language usage. For instance, you may get the impression that some people often use a prepositional phrase – "John gave a book to the students" – while others use the noun phrase – "John gave the students a book". From this you might intuitively draw the conclusion that some people just have a preference for one or the other version, which is a fair interpretation, but not a very scientific one. Now, imagine that you additionally observe that younger people use the noun phrase more often than the prepositional phrase. At this point, you might consider how you can test this, since you still have that linguistics paper you have to write over the summer. With this in mind, you begin thinking about how you can translate your hunch into a research questions, and turn to a corpus. Luckily, the corpus is annotated and you can formulate the necessary search queries, perhaps using regular expressions. You get some reliable counts for both forms, and a set of demographic variables on each speaker. And this is where you run into problems: you can count who uses the two forms how often, but the numbers look somewhat similar. If only there were a test to show you if your hunch is correct, and if there actually is a difference in how different speakers use noun phrases and prepositional phrases. You won't be the least bit surprised when we tell you that there is. In fact, there is more than one. And that we discuss them in this book. In fact, we don't only discuss the tests you can use, but also how to formulate those search queries.

Linguists rely on statistical methods for a whole host of reasons. For one, linguistic differences are often subtle. It is hardly the case that some speaker group, some social class, some genre, etc. uses an expression that others never use. But there are preferences. Consider, for instance, written and language. They are, you know, quite different, but errmm, what are, like, really the differences, like? Many of our written utterances would be acceptable in spoken, and vice versa. With statistics, we can empirically investigate which of these differences are strongest. Or think about language acquisition: a new teaching method claims to be more successful at teaching English than established practices. How can proponents of the new method prove that? By using statistical methods which show that the reduction of errors in their classroom is not only a coincidence. If you are more interested in the language than in the people who speak it, you can use linguistic statistical models to get detailed, data-driven insights into complex interactions, for example into how certain words attract each other in collocations. Alternatively, you may be pondering a philosophical question: how do we even process language? Why is it difficult for us to understand multiple embedded clauses? Why are some zero relatives easy and others difficult for us to parse? When is a sentence a garden path sentence? Well, we veered slightly from the philosophical trajectory, but it will not surprise you to read that statistics can give us a handle to start tackling these psycholinguistic questions. Although the examples here are far from exhaustive, we leave you with a final area where statistics play an especially important role: computational linguistics. Many applications in computational linguistics, from part-of-speech tagging and machine translation, to word-sense disambiguation and information retrieval, use complex statistical methods, and in order to understand how these applications work, you need some understanding of how statistics work.

What all of the issues raised above have in common is that one approach to understanding them is through *the analysis of numerical data*, for instance in terms of frequencies and counts of errors and mistakes when we are dealing with new teaching techniques or in terms of reaction times in psycholinguistics. And therein lies the great affordance of statistics: it is a toolkit with different means of analysing numerical data. The statistical methods are all the more worth learning since they can be used in many different contexts. Once you know, for instance, how significance tests work, any kind of empirical research becomes more transparent. And, of course, you can apply it yourself, on the great swathes of linguistic data which are available in corpora. Indeed, it is no coincidence that quantitative linguistics, which has existed in some form since the late 19th century, rose to its current prominence just as personal computers, these gateways to linguistics data, have become an integral part of daily life. Although the importance of statistics is now widely recognised, it is not yet widely taught. This is a shame, since statstics is a demanding subject, and in the beginning it is good to have a helping hand: perhaps that is why you came here.

And perhaps we can answer some questions that may have arisen.

What does here mean? Here is, of course, the book *Statistics for Linguists*, written by Dr. Gerold Schneider and Max Lauber. The book is based on an e-learning module which Gerold created for the University of Zurich in the mid 2010s. Some of the instructional videos integrated in the book originate from this e-learning course and refer to the slides which accompanied that older course. We have since turned these slides into the text you can read today. If some of the references in the video are not immediately clear, they should become so if you read the surrounding text.

Is this the book for me? We don't know you well enough to presume to answer this, but we can tell you who we made it for. It is a book for someone with either some background or a healthy interest in linguistics, but little theoretical or practical knowledge of statistical methods. And it's a book for someone ready to (metaphorically) get their hands dirty. We discuss fundamental concepts of statistics as well as some more advanced methods in this book, and show you how to implement them in the programming language R. In our examples, we mainly use linguistic data, which we also tend to discuss briefly, but the main focus is clearly not on linguistic theory or insights but on statistical methods.

Will I know everything I need about statistics after reading this? Unfortunately, no. You will get a solid introduction to several important and necessary concepts for understanding statistics, but the book is far from comprehensive. What we do try, however, is to refer you to further material which we find helpful.

How is this book structured? We divided the book into four parts. We begin "The Foundations" by introducing our toolkit R and by discussing the basics of descriptive statistics. In "Inferential Statistics" we introduce two significance tests, the t-test and the chi-square test. We proceed to "Language Models", where we show how to use these statistical methods in the context of two basic language models. Next, we discuss four "Advanced Methods", namely linear and logistic regression analysis, machine learning and topic modelling. Finally, we discuss "The Next Step" required for original research, namely how to phrase search queries to retrieve the phenomena you want to investigate from text corpora.

How should I read this book? There are two things to say about this. Firstly, we recommend reading on a computer which has R installed. This will give you the opportunity to follow our discussions of the different commands and functionalities in R while trying them out yourself. Secondly, we recommend reading the book in the original sequence, especially the "Foundations" and "Inferential Statistics" sections which are written so as to build from one chapter to the next. The "Language Models" and "Advanced Methods" sections probably work somewhat better on their own, but also build strongly on the preceding chapters. This being said, we can only encourage you to use this as a reference work if that is how you benefit most from it.

With that, let's take the first step towards statistics.

Part I

The Foundations

1

IMPORTING AND RETRIEVING CORPUS DATA: FIRST STEPS IN R

In this first part of the book, the Foundations, we want to make sure that everyone has a practical understanding of the programming language R, which we use throughout. In this chapter, you will make your first steps in R and learn how to import data into the program.

What is R?

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=21

The R project page is http://www.r-project.org/. Download the program for your OS from the ETH mirror http://stat.ethz.ch/CRAN/. Install as you usually install programs. Once you have you have installed the R base, you have all the tools you need for the next chapters at your disposal.

Nonetheless, you might want to consider installing RStudio on top of the R base. RStudio is an integrated development environment which allows you to keep track of the data and variables you are using. Moreover, when you are writing a piece of code composed of several commands, the script window in RStudio allows you to break it down into one command per line. If you think these functionalities sound helpful, download RStudio Desktop from https://rstudio.com/products/rstudio/ and install.

R base and RStudio have somewhat different graphic user interfaces, as you can see below. In R base, there is only one window: the console. The console is where you communicate with the program. Here, you enter the commands you want R to execute, and it is where you see the output. In most programming languages, including R, you can recognize the console by the prompt, which looks like this: ">".

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=21

In RStudio, there are four windows. The top left window is the script. Here you can enter multiple lines of code without immediately executing them. The top right window shows the global environment. Here you find all the data you imported and variables have defined. The bottom left window is where plots will be displayed. Finally, and most importantly for now, you find the console in the bottom left window. This corresponds to the R base console in that the commands you enter here will be executed immediately.

In the video tutorials, you see Gerold working with R base. If you are using RStudio, you will want to try what Gerold does in the console.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=21

When you start R, the prompt '>' appears in the console. Select the console window, and type your first command:

```
> 2+2
```

When you hit enter, R executes your command and gives you the output. In this case, the output is:

```
[1] 4
```

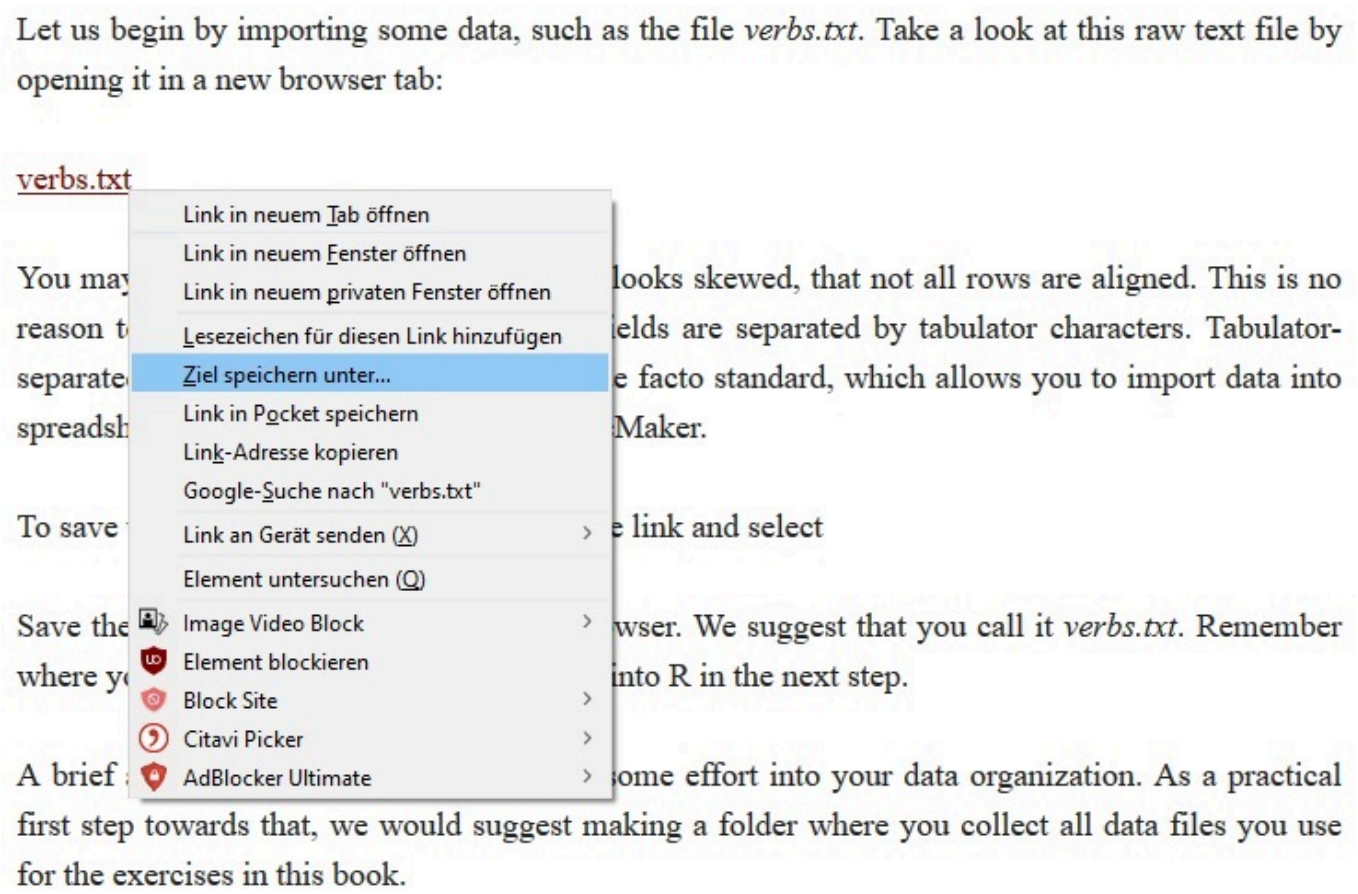
You are probably not surprised. You have just used R as a calculator, and this if a very legitimate way to use it. However, the program is capable of doing much more, and we begin exploring the possibilities in this chapter.

Let us begin by importing some data, such as the file *verbs.txt*. Take a look at this raw text file by opening it in a new browser tab:

verbs.txt

You may have the impression that the table looks skewed, that not all rows are aligned. This is no reason to worry: it is simply because the fields are separated by tabulator characters. Tabulator-separated fields are very flexible, a simple de facto standard, which allows you to import data into spreadsheets like Excel or databases like FileMaker.

To save the file on your PC, right-click on the link and select "Save As...". The image below shows this step in a German version of Firefox (70.0.1):



Save the file as a raw text file from the browser. We suggest calling it *verbs.txt*. Remember where you save it, because you will need to know the file's location in order to import it into R in the next step.

But first, a brief aside: it is generally worth putting some effort into organizing your data and folder structure. As a practical step towards that, we would suggest creating a folder where you store all data files that you download over the course of the book. Of course, you can create subfolders according to chapters or whatnot, depending on your personal preferences. But being somewhat inundated with files ourselves, we cannot overstate the value of having at something of a system.

Returning to R, you can load the *verb.txt* file via the console. When you enter the following command, a dialogue window will open:

```
> verbs <- read.table(file.choose(), header=TRUE, comment.char="", row.names=1)
```

In the dialogue window, navigate to the folder where you stored the file, and open the *verbs.txt* text file. When you open *verbs.txt*, you assign it to the variable verbs. This what the arrow construction, `<-`, effects. The variable *verbs* now contains the table that you opened in the browser earlier.

What is a variable?

In this section, we loosely follow Baayen (2008), chapter 1.

What are we talking about when we say that you assigned the table *verbs.txt* to a variable *verbs*? We are not talking about linguistic variables (not just yet), but about computational variables. In R, variables are objects to which you can assign different values. At its simplest, a variable contains a single value. Consider the following example:

```
> x <- 1 + 2
> 1 + 2 -> x
> x
[1] 3
```

In this example, we assign the outcome of a simple calculation to the variable *x*. And when we open *x*, the output is the outcome of the calculation.

With this in mind, consider again what we did with the *verbs* variable. We assigned a whole table to it, and we can look at it. However, since the whole table is too big to be represented usefully in the R console, let's just take a look at the first few rows. We can do this with the `head()` command:

```
> head(verbs)
  RealizationOfRec  Verb AnimacyOfRec AnimacyOfTheme LengthOfTheme
1               NP  feed      animate      inanimate      2.639057
2               NP  give      animate      inanimate      1.098612
3               NP  give      animate      inanimate      2.564949
4               NP  give      animate      inanimate      1.609438
5               NP offer      animate      inanimate      1.098612
6               NP  give      animate      inanimate      1.386294
```

By default, `head()` shows the first six rows. To view the first 10 rows, we can modify `head()` with the argument `n=10`:

```
> head(verbs, n=10)
   RealizationOfRec  Verb AnimacyOfRec AnimacyOfTheme LengthOfTheme
1                NP  feed      animate      inanimate     2.6390573
2                NP  give      animate      inanimate     1.0986123
3                NP  give      animate      inanimate     2.5649494
4                NP  give      animate      inanimate     1.6094379
5                NP offer      animate      inanimate     1.0986123
6                NP  give      animate      inanimate     1.3862944
7                NP   pay      animate      inanimate     1.3862944
8                NP bring      animate      inanimate     0.0000000
9                NP teach      animate      inanimate     2.3978953
10               NP  give      animate      inanimate     0.6931472
```

Now that we have imported the data into R and taken a first look at it, one question remains unanswered: what does it mean?

How is the data to be interpreted linguistically?

The table *verbs* contains linguistic data on the realization of ditransitive verbs. Each row represents one instance of a ditransitive verb from a corpus. The columns contain information on several aspects of each realization:

RealizationOfRec = Realization of Recipient: is the recipient an NP (as in *I gave **you** a book*) or a PP (as in *I gave a book **to you***)?

Verb: which ditransitive verb is used in this instance?

AnimacyOfRec: is the recipient animate?

AnimacyOfTheme: is the theme animate?

LengthOfTheme: how long is the theme?

This dataset can, for example, be used to find out if certain verbs have a preference for realizing the recipient as an noun phrase (NP) or a prepositional phrase (PP). But before we dig deeper into how we can evaluate data like this with statistical methods, which we will do in later chapters, let's look in some more detail at our variable *verbs* as an object in R.

As you know, *verbs* is a table. In R, tables are referred to as data frames. In this book, we talk about both tables and data frames, in both cases to referring to structured data arranged by rows and columns. The differences is that we use tables generally, and data frames only when we talk about the objects in R.

When we are handling a data frame in R, it is possible to view specific values:

```
> verbs[1,]   # displays row 1
> verbs[,1]   # displays column 1
```

When displaying an entire row, the output shows the value in each column of that row. When displaying an entire column, the output shows the value in each row of that column.

If you use the square brackets to take a look at each of the five columns, you will see that the output of the first four columns conatins an additional line after the contents of the 903 rows. This line is titled `Levels`, and it contains a list of the categories in these columns. In other words, the levels are all the different values the cells in a column take. In this line we see that the *RealizationOfRec* can be either a "NP" or a "PP", and we see that there are 65 verbs, from "accord" to "wish".

You can also view the levels with the command `levels()`:

```
> levels(verbs[,2])
 [1] "accord"    "allocate"  "allow"     "assess"    "assure"    "award"     "bequeath"  "be
 [9] "bring"     "carry"     "cede"      "charge"    "cost"      "deal"      "deliver"   "de
[17] "do"        "extend"    "feed"      "fine"      "funnel"    "get"       "give"      "gr
[25] "guarantee" "hand"      "hand_over" "issue"     "lease"     "leave"     "lend"      "lo
[33] "mail"      "make"      "net"       "offer"     "owe"       "pay"       "pay_back"  "pe
[41] "prepay"    "present"   "promise"   "read"      "refuse"    "reimburse" "repay"     "re
[49] "run"       "sell"      "sell_back" "sell_off"  "send"      "serve"     "show"      "sl
[57] "submit"    "supply"    "teach"     "tell"      "tender"    "trade"     "vote"      "wi
[65] "wish"
```

Here you see, for instance, all of the 65 verbs. Looking at levels like this comes in especially handy if you are not certain what your dataset contains in detail, for instance when you download something in the context of a textbook, and want to see whether it features that one verb you are interested in.

If you look at the last column, you will see that R does not give you a `Levels` line in the output. This is because numerical data is normally interpreted as such, and accordingly is not structured in levels the way categorical data is.

Continuing the work with the square bracket queries, we can look up the value of an specific cell. For instance, we can look at row 401 in the *AnimacyOfRec* column using the following command:

```
> verbs[401,3]
[1] animate
Levels: animate inanimate
```

We can also formulate more complex queries and search, for instance, using ranges. In R, we express ranges with a colon, as in 2:5. Consider the following search query:

```
> verbs[2:5,1:3]
  RealizationOfRec  Verb AnimacyOfRec
2               NP  give      animate
3               NP  give      animate
```

```
4               NP  give      animate
5               NP offer      animate
```

You see that with these range searches, R displays rows 2 through 5 and columns 1 through 3.

Alternatively, we can restrict our searches to small selections. We do this by using the command `c()`. This command combines values into a vector or list. You could also think of `c()` as a command to concatenate.

Say we want to look at rows 1 to 6, but for some reason we really don't want the third row to be displayed. We can do this using `c()`:

```
> verbs[c(1,2,4,5,6),]
```

We can, of course, do the same thing for columns:

```
> verbs[,c(1,3)]
```

All of the search queries we looked at so far are concerned with how to access certain positions in a data frame. But usually, we are less interested in seeing a specific cell than looking at certain values.

How can we formulate queries for specific values?

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=21

When the contents of a cell (or several cells) interests us more than its position, we can formulate something like a database query. For instance, we may be interested in seeing all rows in *verbs* which have an animate theme. In this case we would type the following command:

```
> verbs[verbs$AnimacyOfTheme == "animate",]
    RealizationOfRec  Verb AnimacyOfRec AnimacyOfTheme LengthOfTheme
58                NP  give      animate        animate     1.0986123
100               NP  give      animate        animate     2.8903718
143               NP  give    inanimate        animate     2.6390573
390               NP  lend      animate        animate     0.6931472
506               NP  give      animate        animate     1.9459101
736               PP trade      animate        animate     1.6094379
```

This query looks more complex than anything we have seen so far, so let's break it down. As before, we have *verbs* followed by square brackets. This means that we want to access the content of the variable *verbs* as defined by the criterion in the square brackets. In the square brackets we have `verbs$AnimacyOfTheme` which means that within the *verbs* variable, we want only those cases in which the animacy of theme is animate. The two equal signs are used to test for exact equality, and this is what gives us the desired output.[1] This type of query takes some getting used to, but it is how R allows us to do something like a database query.

With this understanding, you can easily formulate analogous queries:

```
> verbs[verbs$Verb == "lend",]
```

Of course, these can be expanded to include multiple elements:

```
> verbs[verbs$Verb=="lend" | verbs$Verb=="sell",]
```

The "|" means "or", so R displays all the rows containing either the verb "sell" or "lend".

Now, of course, our data frame has multiple columns of categorical data, so we can use this syntax to search for rows which are consistent with two conditions:

```
> verbs[verbs$Verb=="lend" & verbs$RealizationOfRec=="PP",]
```

1. For a definition of all operators, see "Chapter 3: Evaluation of Expressions" in the *R Language Definition* manual, which is available at https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf

As you would expect the "&" means "and", wherefore the output contains those rows in which the recipient of the word lend is realized as PP.

That's it! You have seen how to import data and phrase search queries, and you will learn more about R in the next chapter, Introduction to R.

Reference:

Baayen, R. H. (2008) *Analyzing Linguistic Data. A Practical Introduction to Statistics Using R*. Cambridge University Press, Cambridge.

2

INTRODUCTION TO R

In this chapter, we will take a step back and discuss several defining features of R. If you are not yet familiar with any programming language, you will learn the conventional terminology and get to know practical and common concepts of programming.

Why R?

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

In the last chapter, you saw how R can be used to get a handle on large data sets, and if you got this far, there will be no need to recapitulate how statistics can augment linguistic research. If, however, you already know another programming language you might wonder why you should learn R. There are several reasons:

- It's very close to statistics
- It's powerful, as you will see later on
- It makes minimal use of procedural style and has a functional flavor
- There are many libraries for specific tasks
- The code is often less hackish than e.g. Perl
- It has less syntactic overhead than e.g. Perl
- It is less verbose than e.g. Java
- It is conceptually easier than Prolog

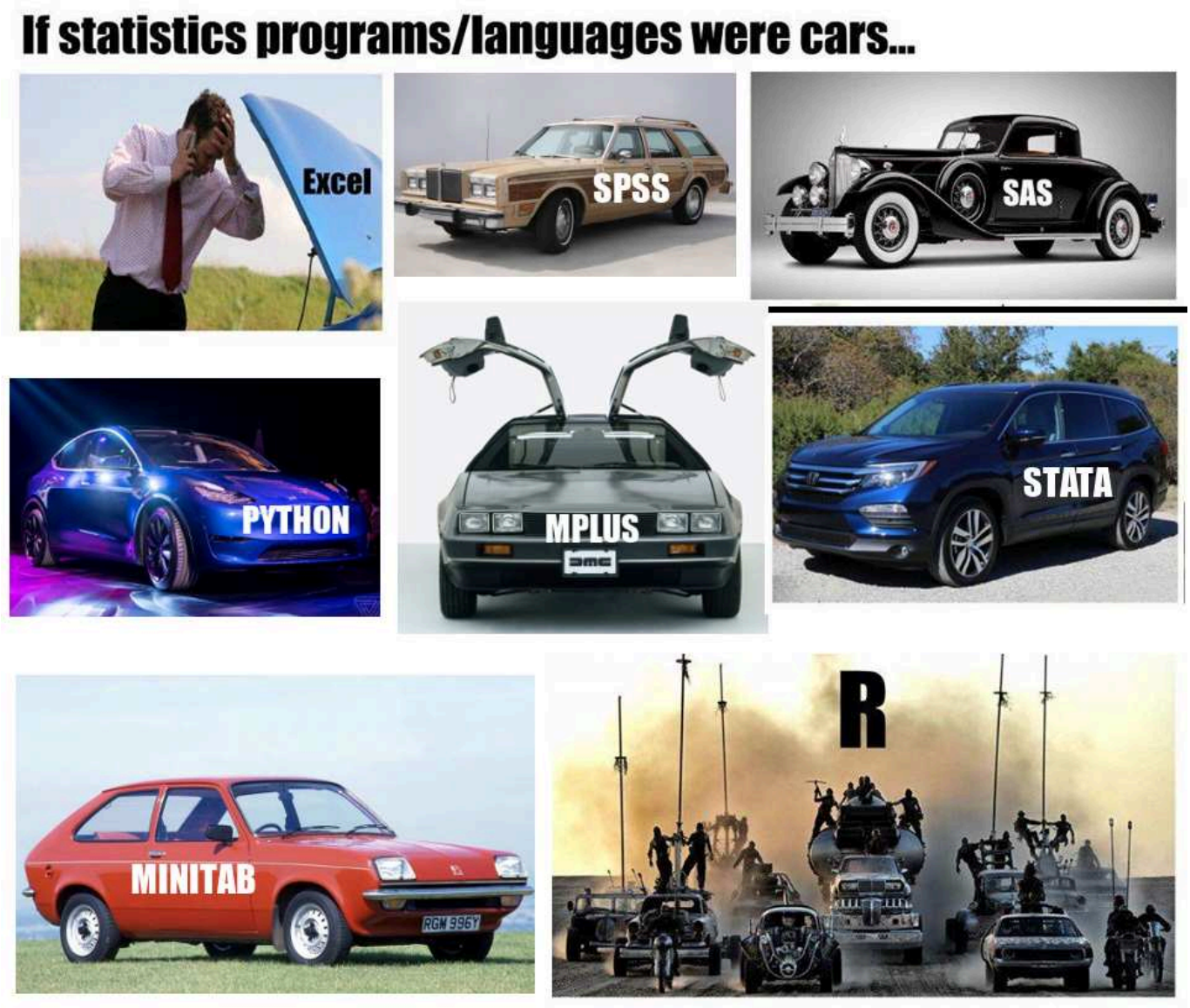
And, of course, different languages are useful for different purposes. Don't use R instead of, but in addition to the languages you already know. There are, for example, APIs from Perl, Prolog and Python to R which allow you to integrate the different languages with each other.

Unlike other statistics programs, R is a free, open source language, and has a very active community committed to programming extensions to R. These extensions are known as library, and usually if you have a clear idea of what you want to do in R but can't find a predefined function to do it, you will find that someone has programmed a library which serves your need. We are grateful to guyjantic on Imgur for visualizing the allure of R over other programs:

Introduction to R

This section builds on Gries (2008), chapter 2.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23



2.1 R in comparison (https://imgur.com/gallery/CCr9jE9).

In the last chapter, you already got a taste of some simple commands. Here, we briefly recapitulate some of what we discussed in the previous chapter and introduce several more foundational concepts.

For those of you new to programming, it is useful to think of programming as writing sets of calculations and/or commands. These commands have to be entered at the prompt in the console of R. Remember that the prompt is the >.

At the most basic level, we can use R as a calculator. In those cases, we just enter our calculations and get the result:

```
> 17/2
[1] 8.5
```

As we have seen, we can also define variables. Variables can take many forms. Most frequently, though, our variables are strings of characters, which are identifiable by the quotation marks around them; numeric values, either single numbers or vectors containing multiple numbers; or data frames, which is to say tables.

Here you see an example of a variable which contains a string and of one which contains a numeric vector:

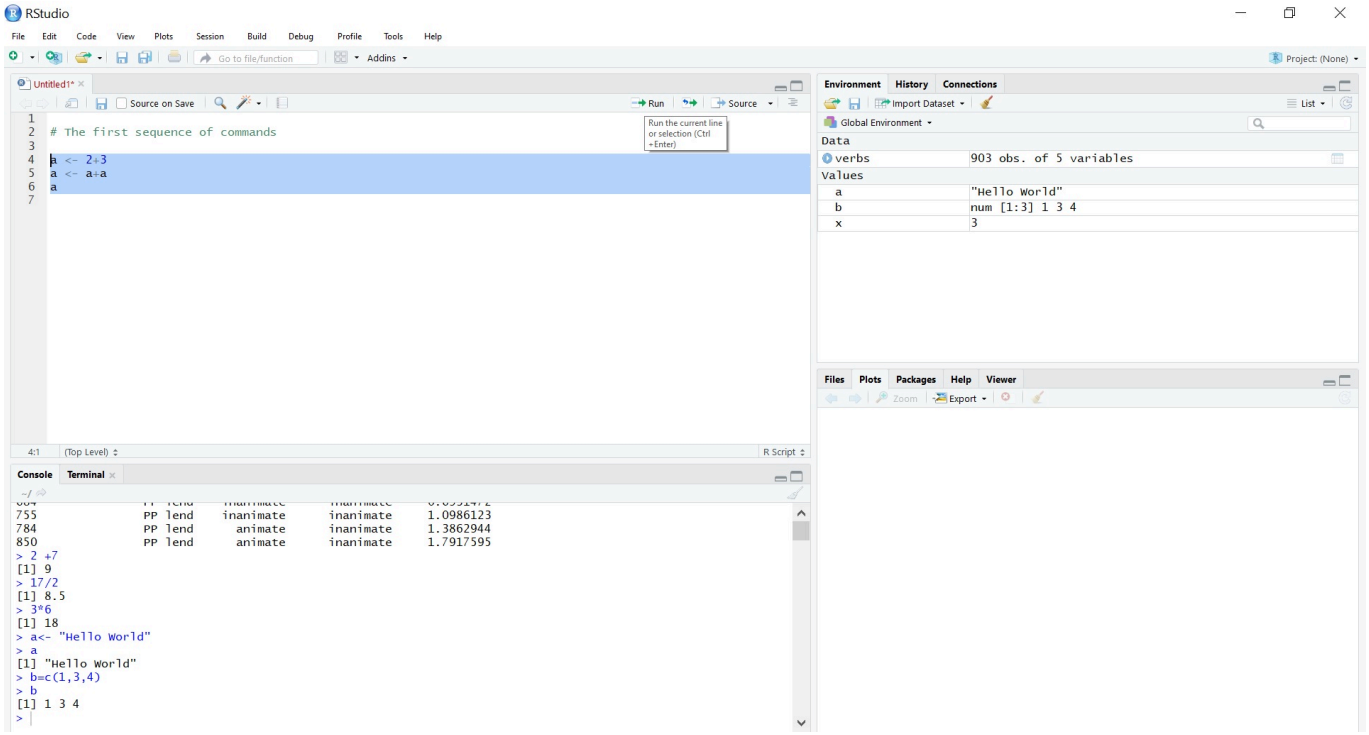
```
> a<- "Hello World"
> a
[1] "Hello World"
> b=c(1,3,4)
```

```
> b
[1] 1 3 4
```

So far, we have only shown examples where there is one command per line. However, it is also possible to enter sequences of commands. To do this, we enter several commands separated by a semicolon:

```
> a <- 2+3 ; a <- a+a ; a
```

In RStudio, we can also use the script to write and execute a sequence of commands. When working in the script window of RStudio, you can write each command on a new line. Then you select the lines you want to run and execute them by pressing "Ctrl+Enter" or clicking "Run" in the top right of the window.



2.2. A sequence of commands in the RStudio script

As we mentioned above, R contains a lot of predefined functions beyond simple mathematical ones. Unless you use R only as a calculator, you will use functions which take an argument. This means that function is performed on an object, like one of our variables. Let's begin by looking at a simple mathematical function which takes an argument: the square root.

We can reassign our variable *b* to become the square root of 5 and then print the result:

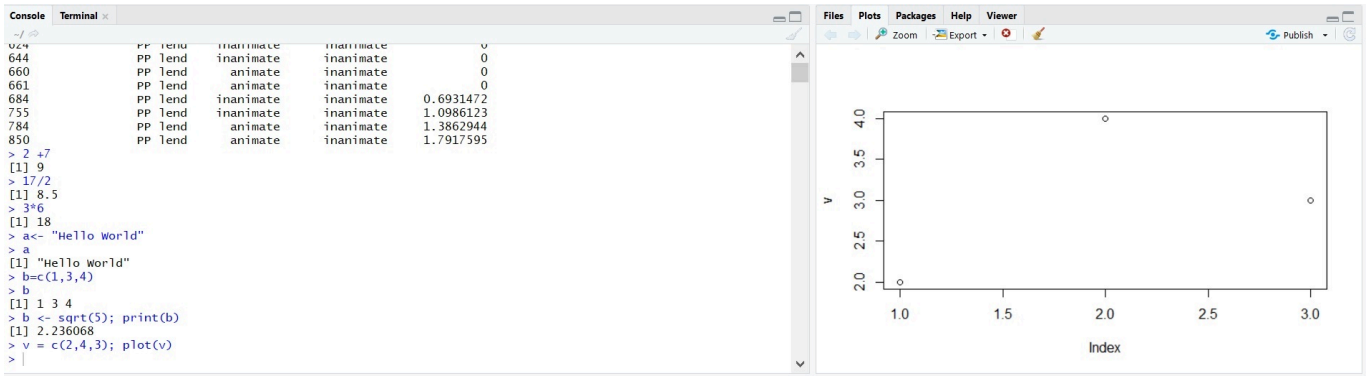
```
> b <- sqrt(5); print(b)
[1] 2.236068
```

Or we can assign a numeric vector to the new variable *v* and plot the result. The picture below contains the two bottom windows of RStudio, showing the console with the command and the resulting plot.

A further function which takes an argument is `sample()`. In fact, this function takes two arguments. Test your intuition of how this function works in the exercise below.

> An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

In the last chapter, we already briefly touched on ranges. This is a fairly simple concept, but it can be very useful if you want to generate a bunch of numbers quickly (as sample data, for instance). The operater which enables us to create a



2.3. Plotting vector v in RStudio

range is the ":". In the example below, we use this to generate a variable containing a numerical vector ranging from 0 through 10:

```
> MyRange <- c(0:10) ; MyRange
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

A final basic concept are indexes, which we als touched on in the last chapter. Indexes refer to particular positions in vectors and can be accessed using square brackets. For instance, to access the fifth position in the *MyRange* variable we created above, we can use the command:

```
> MyRange[5]
[1] 4
```

As we discussed at some length in the last chapter, we can also use indexes to access specific cells in data frames. The data frame examples, of course, had two dimensions (rows and columns), while the vector only has one (being a list). If you try to incorporate a second dimension in *MyRange* by inserting a comma into the index, you get an error message:

```
> MyRange[5,]
Error in MyRange[5, ] : incorrect number of dimensions
```

So far, so good, or so we hope. As you might expect, things get more complicated fairly quickly. For instance, we can formulate conditional commands in R, meaning that the output depends on an element which is determined by a step in a squence of commands, rather than by us.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

Take, for example, the sequence of commands below:

```
> randomthingy = sample(c(1,2,3),1) ; if (randomthingy > 1) {print ("You are lucky")} else {
[1] "You are lucky"
```

You are now, of course, familiar with the `sample()` function, so you will immediately see that the first command in this sequence assigns the value 1, 2 or 3 to the variable randomthingy. The second part of the sequence is a little more complicated: it is a logical test. The argument in brackets behind the `if`, (randomthingy > 1), test whether the number we sampled before is bigger than one. If this is the case, the output is "You are lucky". If, however, the number we sampled is equal to one, the output is "I am sorry".

You can see that the conditional output is put in quotation marks, because we want the output to be strings, and in

curly brackets, [ ]. The curly brackets are referred to as blocks. In this example, the blocks allow us to conditionally define the output.

Blocks are also used in another important feature of programming: loops. Loops allow you to run multiple iterations of a command. Before we discuss what this means in detail, take a look at this example:

```
> for (i in c(1:3)) {myS = sample(c(1,2,3),i) ; print(myS)}
[1] 2
[1] 2 1
[1] 1 2 3
```

What is going on here? This is perhaps easiest to grasp by looking at the what is in the block. You can see that it is a squence of familiar commands: in the first, we draw a sample from a vector and assign it to the variable *myS*, and in the second we tell R to display the variable.

The new element here is the second argument in the `sample()` function, *i*. What is *i*? If you look at the `for()` command in front of the block, you get the answer. Here, we tell R to iterate through the vector ranging from 1 through 3. This means that in the first iteration, *i* takes the value 1. Then, the commands in the block are executed. In the second iteration, *i* takes the value 2. Again, the commands in the block are executed. In the third and final iteration, *i* takes the value 3. Again, the commands in the block are executed.

If it was slightly repetitive to read the explanation, you have understood why loops are a useful concept. Imagine you want to run a loop through not three, but, say, 10'000 iterations. With this small but powerful line of code, you can sometimes save yourself hours of tedious manual labor.

Often it is not entirely clear what a command is supposed to do, and there are many default assumptions being made (by R) in the background. Run the following loop, which is identical to the one above, except that it runs not through three but through eight iterations:

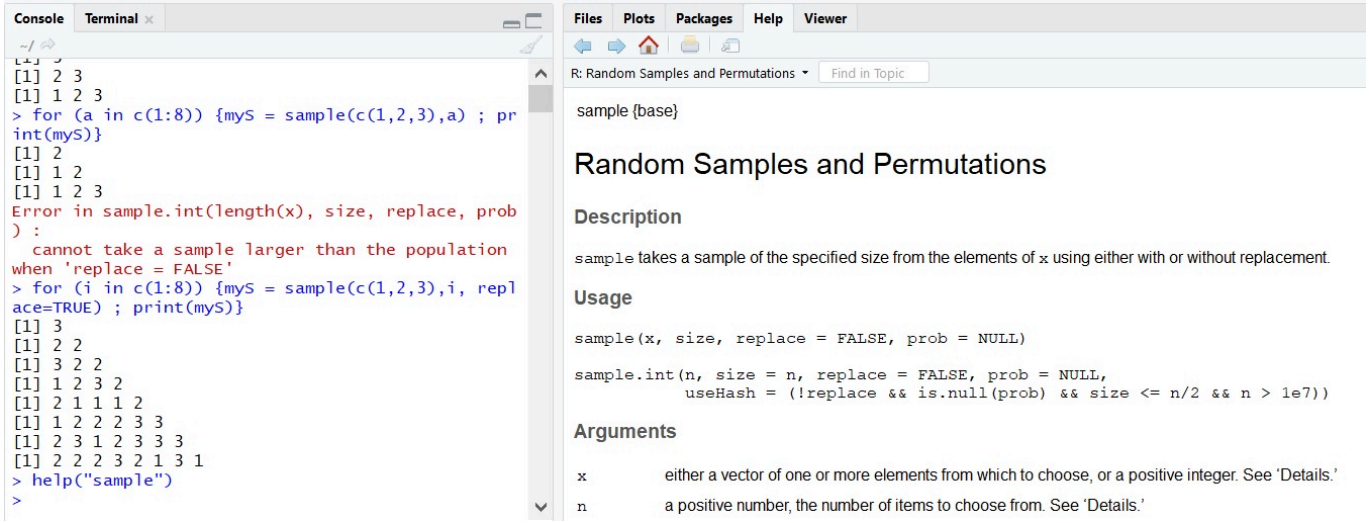
```
> for (i in c(1:8)) {myS = sample(c(1,2,3),i) ; print(myS)}
[1] 2
[1] 1 2
[1] 1 2 3
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
```

The error message R prints shows that we are working with the default assumption under which the `sample()` function samples without replacement. In other words, there are only three values which can be drawn, and once all three numbers are drawn R prints the error message because it cannot draw the fourth value which would be required in the fourth iteration. In order to avoid this error, we can augment our sample command with a further argument, where we define "replace = TRUE":

```
> for (i in c(1:8)) {myS = sample(c(1,2,3),i, replace=TRUE) ; print(myS)}
[1] 3
[1] 2 2
[1] 3 2 2
[1] 1 2 3 2
[1] 2 1 1 1 2
[1] 1 2 2 2 3 3
[1] 2 3 1 2 3 3 3
[1] 2 2 2 3 2 1 3 1
```

With this additional argument, we allow for sampling with replacement, which means that R can draw each value multiple times. Now the loop runs without error messages.

It is not always obvious which functions take which arguments, let alone what the default assumptions of these arguments are. In those cases, you can look up the default arguments using R's inbuilt documentation using one of the two help commands: `help(sample)` or `?sample`. If you are using the R base, a new window will open with the documentation file, and if you are using RStudio, the documentation will open in the bottom right window.



2.4. Screenshot of the documentation to sample() in RStudio

In the beginning, the information in the documentation may be more confusing than helpful because of the style it is presented in. Most help pages feature several examples at the end of the page, and it is worth scrolling down to look at those if you get stuck. In time, you will become used to the information in the help pages and understand it readily.

A final concept we want to introduce is nestedness. Nestedness describes situations where commands contain other commands, leading to a layering of commands. Look, for instance, at the following example:

```
> sort(sample(c(1,2,3),5, replace=T))
[1] 1 1 2 3 3
> sort(sample(c(1,2,3),5, replace=T))
[1] 1 1 2 2 3
```

Let's begin again by looking at the familiar elements. In brackets, you see a sample function which draws 5 observations with replacement from a vector ranging from 1 through 3. We discussed similar examples above, and there we always received an unsorted sample. Here, however, the numbers are sorted in ascending order. You are correct in thinking that this is because of the `sort()` function. What may be less evident is that we are dealing with nested commands here. You can clearly see this if you perform the steps separately:

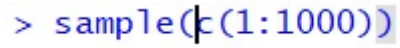
```
> chaos <- sample(c(1,2,3),5, replace=T)
> chaos
[1] 3 1 3 3 3
> order <- sort(chaos)
> order
[1] 1 3 3 3 3
```

With an example like this, nestedness is fairly easy to see and to work with. However, consider this more complex set of nested elements:

```
> L <- sample(c(1:10),1); if (L == 1) {print ("O dear ...")} else { if (L > 8) {print("you l
```

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

It is with examples like this that you will start to get a feeling for the importance of keeping a good overview over your brackets. In the base R, you are responsible for getting the brackets right yourself, while in RStudio offers a bit of assistance on that front by coloring the other half of the bracket grey if your text cursor touches a bracket:

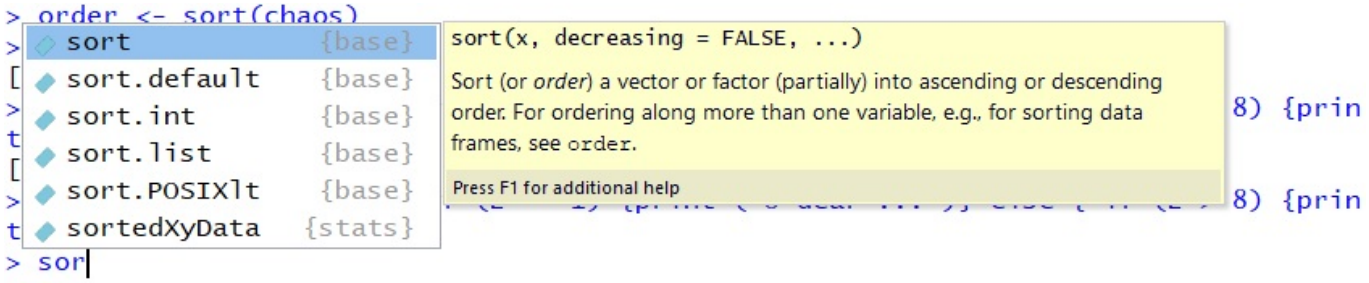


5. RStudio's grey bracket assistance

A user-friendly editing function which is included in RStudio as well as the base R is the history. You can scroll through the history with the "arrow-up" key, which saves you the trouble of copying and pasting if you want to run a command multiple times without (or with few) adjustments. If you scroll back too far, you can use the "arrow-down" key to get to where you want to go. The history allows you to retrieve any command you entered during the current session.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

Another user-friendly editing function included in R is the auto-fill. If you type the letters "sor" into the console and hit the TAB key, you will see a small window open which contains all possible completions of "sor", from `sort` to `sortedXyData`. Use the up- and down-arrow keys to navigate through your options and hit TAB again on the one you want.



6. The auto-fill function in RStudio

Armed with these concepts, you should be prepared for most of the things we discuss from here on.

Mistakes and the importance of practice

In the beginning, you will make a lot of mistakes. This does not make you special. From beginners to experienced programmers, everyone makes mistakes and runs into error messages from time to time. In fact, the more time you spend programming, the more error messages you will see. In a sense, learning a programming language forces you to confront your mistakes more than other solitary practices, since computers are extremely nit-picking, and take everything you enter exactly as it is, regardless of whether it makes sense or not. To help you avoid more error messages than are necessary, we compiled a list of several very frequent mistakes. In fact, we let you sort out the correct from the incorrect commands.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

An interactive or media element has been excluded from this version of the text. You can view it online here:
https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

You will spend a lot of time browsing manuals, debugging and googling. As the saying goes, there's no learning like learning the hard way. While this is perhaps a negative way of looking at things, it has a decisively positive flipside: you are in constant interaction with the computer. So you get error messages, and you get output, and the more you practice, the more often the output will be what you want it to be and the rarer the error messages become (although if you stop getting any error messages, it means that you probably stopped programming).

What is more, you are not alone in this. The online community has answered a lot of questions, and often it helps to just google whatever you want to know. For instance, you could search Google for "R random number generate" and find many different solutions on how to do that. We should also add, though, that it becomes easier to find solutions online if you are familiar with the R terminology, so we would encourage you to think, for example, about "data frames" rather than "tables" when working with R. We cannot practice for you, but we make sure that you pick up the correct terminology from reading this book.

First mistakes

An interactive or media element has been excluded from this version of the text. You can view it online here:
https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

After our litany on practicing, we want to give one more piece of advice before beginning with statistics proper. One of the difficulties beginners encounter, and indeed some of the most frustrating challenges occur when importing files into R. We already discussed one way of doing that in the previous chapter, where you imported a tab-separated table with headers into R. In addition to tab-separated files, there are two more structures which are worth knowing. Open these two raw text files in new tabs:

vector1.txt and vector2.txt

First of all, since they are one-dimensional we can identify both of these files as vectors. Then, we see that the data points in the first vector are separated with new lines, and those in the second are separated by a whitespace. When importing a file it is very helpful to be aware of how the file is structured.

For instance, if we import *vector1.txt* using the command we file.choose() that we saw in the previous chapter, we can use a simple piece of code:

```
> v1 <- scan(file=choose.files())
```

Again, a window will open and we can open *vector1.txt* from whereever we saved it. However, if you try to use the same code to open *vector2.txt* you get an error message:

```
> v2 <- scan(file=choose.files())
Error in scan(file = choose.files()) :
  scan() expected 'a real', got 'anton'
```

The error message occurs for two reasons. Firstly, *vector2.txt* contains character strings, not numbers. Secondly, the data points are separated, as we said, by a whitespace. We have to augment the `scan()` command by two arguments:

```
> v2 <- scan(file=choose.files(),sep=" ",what = "char"); v2
Read 5 items
[1] "anton"  "berta"  "caesar" "dora"   "emil"
```

These arguments, which are also called flags, are not optional for the separator and the data type. If you don't include them, you cannot open *vector2.txt*.

When working on a script in R over multiple sessions, it becomes tedious to manually select the file you work with each time. In order to target the correct file automatically, adapt the file path and enter the absolute file location:

```
> v2 <- scan(file="C:/Users/Max/Documents/R/stats for linguistics/chapter 2/vector2.txt", wh
Read 5 items
```

This also works with the `read.table()` function which we used in the previous chapter:

```
> verbs <- read.table(file="C:/Users/Max/Documents/R/stats for linguistics/chapter 1/verbs.t
```

So far, we have only discussed how to get data into R. But, of course, sometimes we want to edit the data in R and then save the results. For this we can use the `cat()` function.

Imagine, for instance, that you want to create a list of possible names for your unborn child. So far, you have compiled a list of of alphabetically sorted names: the *v2* you imported above. You feel that five names just don't cut it, you want at least seven. So you add the two next candidates to the list:

```
> v2 <- append(v2, c("friedrich", "gudrun"))
```

In a next step, you want to save the new variable with seven names so you can print the list and discuss the new additions with your spouse. To do so, you need the `cat()` function and the arguments it takes: the variable you want to save, the file path and the separator type:

```
> cat(v2, file="C:/Users/Max/Documents/R/stats for linguistics/chapter 2/vector2_more_names.
```

Here, we decided to save the longer list as a new file, with line breaks separating the names. If you want to overwrite an existing file, you can also use the `file.choose()` function:

```
> cat(v2, file.choose())
```

> An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=23

Vectors

Finally, since we are already working with them, let us add some words about vectors. We have discussed how vectors can be imported, lengthened and saved. With *v1* and *v2* we have seen that we can store both numeric and character data in vectors. In the example above, we played around with *v2*, so now let's take a look at what we can do with the numeric vector, *v1*.

Like we did above, we can add more information to the vector:

```
> numbers <- append(v1,c(8,9,11)) ; numbers
[1]  1  2  3  4  5  8  9 11
```

Then we can calculate the mean:

```
> mean(numbers)
[1] 5.375
```

Or run it through a loop:

```
> for (n in c(1:10)) {numbers = append((mean(numbers)+length(numbers)),numbers)} ; numbers
 [1] 30.65330 28.71213 26.77463 24.84129 22.91272 20.98965 19.07298 17.16389 15.26389 13.375
[12]  2.00000  3.00000  4.00000  5.00000  8.00000  9.00000 11.00000
```

There are many ways to manipulate numeric vectors, and we will discuss them more in subsequent chapters.

What happens if we encounter a vector which contains both numbers and words? We can try that out by generating one:

```
> a <- c(1,2,3, "hi!")
```

First, let's check whether this is even a vector:

```
> is.vector(a)
[1] TRUE
```

We can also check out whether *a* has as many positions as we think it does:

```
> length(a)
[1] 4
```

Then, we can use the `str()` function to view the structure of this object *a*:

```
> str(a)
 chr [1:4] "1" "2" "3" "hi!"
```

Here, we see that even the numbers in this vector are characters. This makes sense since R is incapable of coercing characters strings into numbers:

```
> as.integer(a)
[1]  1  2  3 NA
Warning message:
NAs introduced by coercion
```

And since numbers can be converted into characters, this is what R does.

In the next couple of chapters, we will discuss the fundamentals of statistics, which means we will be working mostly with numeric data. You can find out what this means in detail by continuing to the next chapter, Basics of Descriptive Statistics.

If you are interested in acquainting yourself with more useful functions in R before continuing, we recommend reading the documentation of and playing around with the following functions:

- as.characters
- as.integers
- attach
- plot
- hist
- barplot
- pie

For further reading and more exercises we recommend reading chapter 2 in Gries (2008) and visiting his website at http://www.stgries.info/research/sflwr/sflwr.html.

Reference

Gries, Stefan Th. 2008. *Statistik für Sprachwissenschaftler*. Göttingen: Vandenhoeck & Ruprecht.

3

BASICS OF DESCRIPTIVE STATISTICS

In this chapter, we discuss the basics of descriptive statistics and show you how to calculate the most common descriptive statistical measures in R.

Beginning Statistics

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

Where in programming the main way to make progress is trial and error, when it comes to learning descriptive statistics, we recommend reading different definitions of the main concepts until you understand them. In a first instance, we will provide the best explanations that we can come up with. If questions remain, or if you want to dive deeper into the topics, we can recommend starting with the following texts:

- Woods, Anthony, Paul Fletcher and Arthur Hughes. 1986. *Statistics in Language Studies*. Cambridge: CUP.
- Oakes, Michael P. 1998. *Statistics for Corpus Linguistics*. Edinburgh: EUP.
- Butler, Christopher. 1985. *Statistics in Linguistics*. Oxford: Blackwell.

Consider these three texts the tip of the iceberg: there are many introductions statistics, and if neither our book nor any of our recommendations manage to spark your understanding of statistical concepts, you are bound to find some other book which does.

Average: Mean, Median, Mode

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

Let's jump right in. Statistics, as we mentioned in the introduction, provide many different ways of describing and analysing numerical data. The easiest statistical measure is the average. The most complicated thing about the average is that there are three different measures that can correspond to what is colloquially referred to as the "average".

The most common measure of the average is the mean. The mean of (1, 5, 6) is 4. It is calculated by summing all of the numbers you are interested in and dividing the resulting value through the amount of numbers. Mathematically, the mean $\bar{x}$ is formulated like this:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The sum operator, $\sum$, is a loop which runs through all observations, x_i, and calculates the sum of all values. In our example, we have three observations: $x_1 = 1, x_2 = 5, x_3 = 6$. If we sum these up, the result is 12. This value is then divided by the number of observations, in our case n=3. Thus, we arrive at a mean=4.

The median is the second measure which is sometime referred to as the average. The median is the value which lies in the middle of a sorted list of numbers. The median of (1,2, 6) is 2. For every list, 50% of the observations have values higher and 50% have values lower than the median. If a list has an even number of entries, the median is defined as the mean of the two numbers in the middle of the list.

The final measure for the average is the mode. The mode is the value which is most frequent in a list. The mode of (1,1,4,6) is 1. In distribution graphs, the mode is the peak.

Averages in R

We can easily calculate the three measures for the average, mean, median and mode, in R. To do this, let us assign create a vector *x* containing a list of numbers:

```
> x <- c(10.4, 5.6, 3.1, 6.4, 6.4, 21.7)
```

Once we have defined the variable x, we can use the predefined mean function in R:

```
> mean(x)
[1] 8.933333
```

The same is true for the median:

```
> median(x)
[1] 6.4
```

There is no direct way to calculate the mode, but that does not mean that it cannot be done. It just takes more steps do so. First, we need to create a table:

```
> table(x)
x
 3.1  5.6  6.4 10.4 21.7
   1    1    2    1    1
```

This command shows us which value occurs how often. With the function `max()`, we can calculate the maximum:

```
> max(table(x))
[1] 2
```

In words, this means that the element which occurs most frequently occurs twice. Finally, we can use a closely related command, `which.max()`, to identify which value it is that occurs most:

```
> which.max(table(x))
6.4
  3
```

As we could already read from the table we created earlier, the mode is 6.4, which stands in the third column of the table. When working with a small set of observations, like we are here, it is possible to just use the table function and then look for the value which occurs most frequently. However, once you start working with thousands of observations, this is no longer possible, so it is worth knowing where to look up the steps to calculating the mode.

Calculate the mean and the median in R without using the predefined functions `mean()` and `median()`. Try it out in R and compare your solution to the example solutions below.

> An interactive or media element has been excluded from this version of the text. You can view it online here:
> https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

Apart from the fact that we will need some of these measures for the average later on, they are useful in their own right. Most of the time when you are presenting any sort of quantitative finding, you will want to summarize your data effectively using these measures.[1]

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

We mentioned distribution graphs above, in the context of the mode. Distribution graphs are a very useful means of representing continuous numeric variables, such as:

- Marks of a class
- The weight of UK citizens (e.g. a sample of 1000 arbitrarily chosen people)
- Frequencies of "has" in the 44 texts of LOB Category A (Press: reviews)
- Word lengths (in letters) in 6 "London Gazettes" of 1691, as you see here:

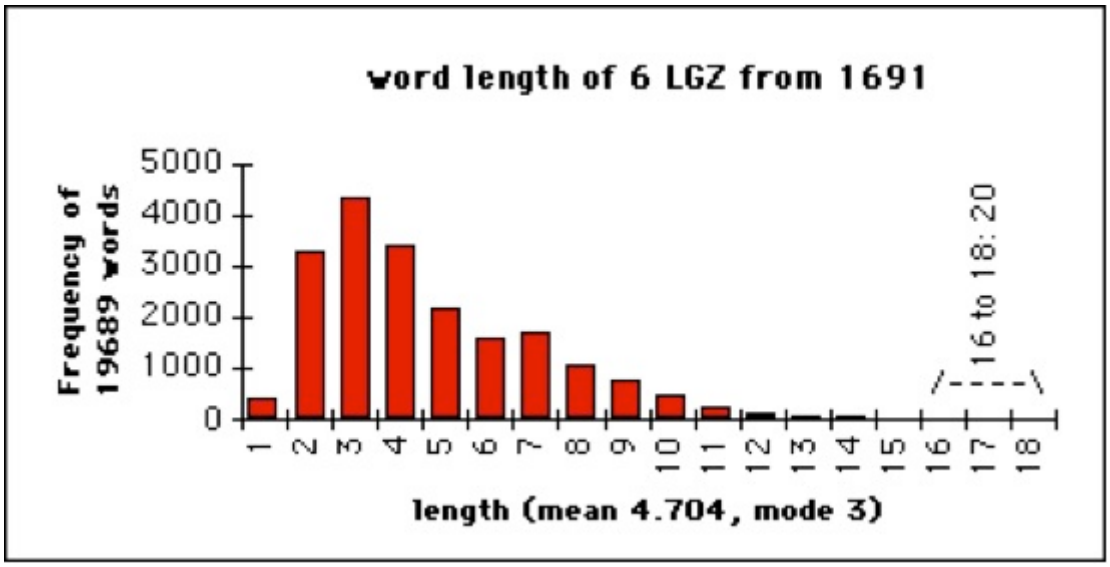


Figure 3.1: Word length of 6 London Gazettes from 1691

So what do these distribution graphs visualize, and how can we make them? Using an example from Woods et al. (1986: 16ff.) we discuss here first the theory, then we put the theory into practice in R.

Let us begin by looking at the raw data:

What you see in the table are exam scores of 108 Cambridge students from June 1980. There are 107 observations, and the data is unsorted. How do we proceed to create a distribution graph for this set of data?

In a first step, we have to sort the numbers, and, unless there is a really good argument for the opposite, we do this in ascending order. In a second, and more important step, we interval classes like these:

Each interval needs to contain the same difference between the upper and the lower bound. In the image above, we see that each interval has a length of 15 points. The first interval contains scores between 110 and 124, the second interval contains scores between 125 and 139, and so on. For each of the ten intervals, we count the frequencies. We can easily create a table with the frequencies, and present the information in the form of a distribution graph:

While it is possible to create distribution graphs like this in Excel, it requires either a fair amount manual work or the use of rather complex functions. In R, we can do this much more easily.

The scores are available for download as a raw text file here:

1. One example for this, picked because it is too fitting, is in Lazerton et. al (1987): "Respondents were asked to estimate the number of statistics or research methods courses they had taken. The number of courses ranged from 0 to 12 with a mode of 1, a median of 1.66, and a mean of 2.27 (SD = 2.18)" (267):

194	184	135	161	186	198	150	240	147	174	197
176	183	117	161	185	186	208	200	157	212	191
174	192	145	162	186	148	241	184	201	208	177
135	229	179	208	209	203	145	201	204	192	179
224	209	179	223	192	221	239	238	199	174	145
226	214	211	215	176	238	184	221	198	196	184
192	164	209	142	196	160	165	166	224	229	184
171	163	207	179	197	120	255	150	233	188	175
225	156	211	190	204	222	219	186	160	189	
218	149	160	188	224	140	220	149	170	197	

Table 3.1: Scores of 107 students in June 1980 Cambridge, not summarised

Class interval	Tally	Frequency
110–124	||	2
125–139	||	2
140–154	𝍸 𝍸 |	11
155–169	𝍸 𝍸 ||	12
170–184	𝍸 𝍸 𝍸 ||||	19
185–199	𝍸 𝍸 𝍸 𝍸 |||	23
200–214	𝍸 𝍸 𝍸 ||	17
215–229	𝍸 𝍸 𝍸	15
230–244	𝍸 |	6
245–259	|	1
		108

Table 3.2: Frequency table of the scores of 108 students in June 1980 Cambridge, summarised

cambridge-scores.txt

We discussed in the previous chapter how to open files in R. You will remember that you can open files using the file path:

```
> scores <- scan(file="/Users/gschneid/Lehre/Statistics/cambridge-scores.txt", sep="\n")
```

or by opening a search window:

```
> scores <- scan(file=file.choose(), sep="\n")
```

Once the data is assigned to the variable *scores*, we can easily create a histogram using the predefined `hist()` function:

```
> hist(scores)
```

This should result in this histogram:

The chart we get here more or less resembles something called the normal distribution (see image 3.4 below). In many other cases, you will not get normally distributed data. Regardless of what your intuition is about the distribution of your data, it is worth plotting it in a histogram (or another distribution graph) to see how your data is distributed. This is especially important because many statistical methods are adapted to particular types of data distribution. For the

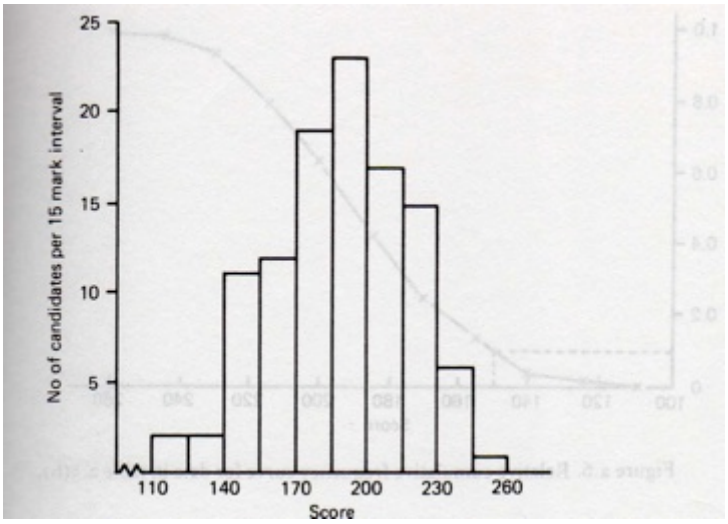


Figure 3.2: Histogram of the frequency data in table 3.2

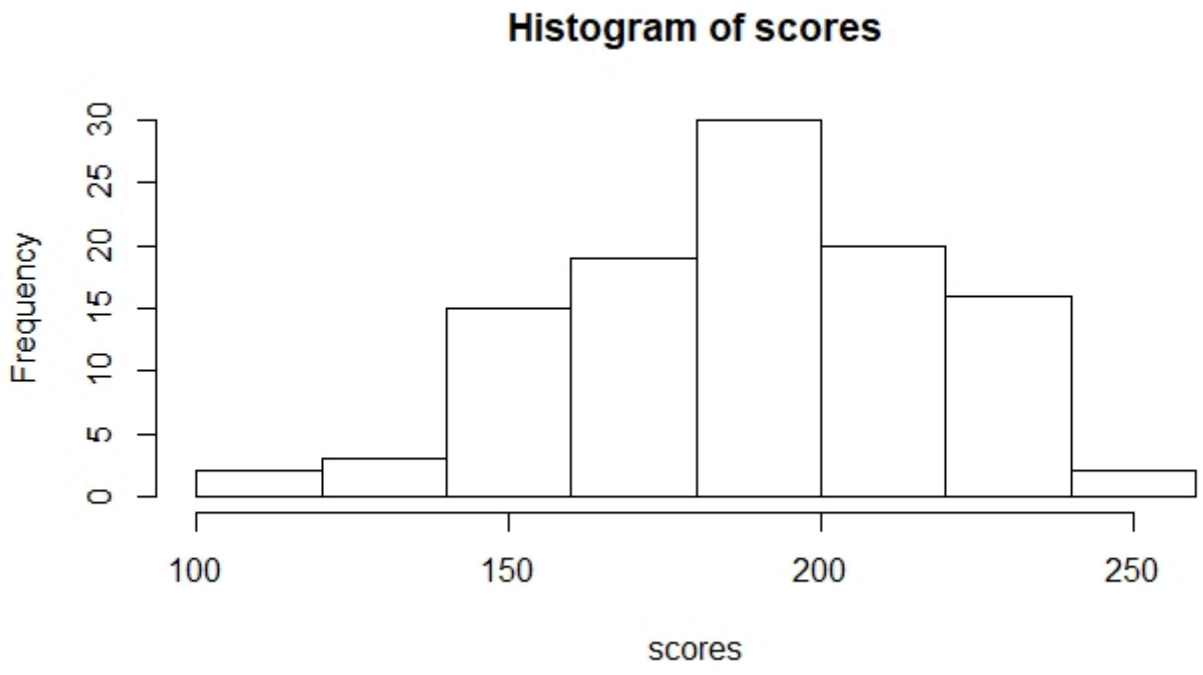


Figure 3.3: Histogram of the Cambridge scores in R

moment, it is enough to know that the normal distribution is frequent and very important in statistics, which is why we discuss it at greater lengths later on.

Dispersion: Variance, Standard Deviation

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

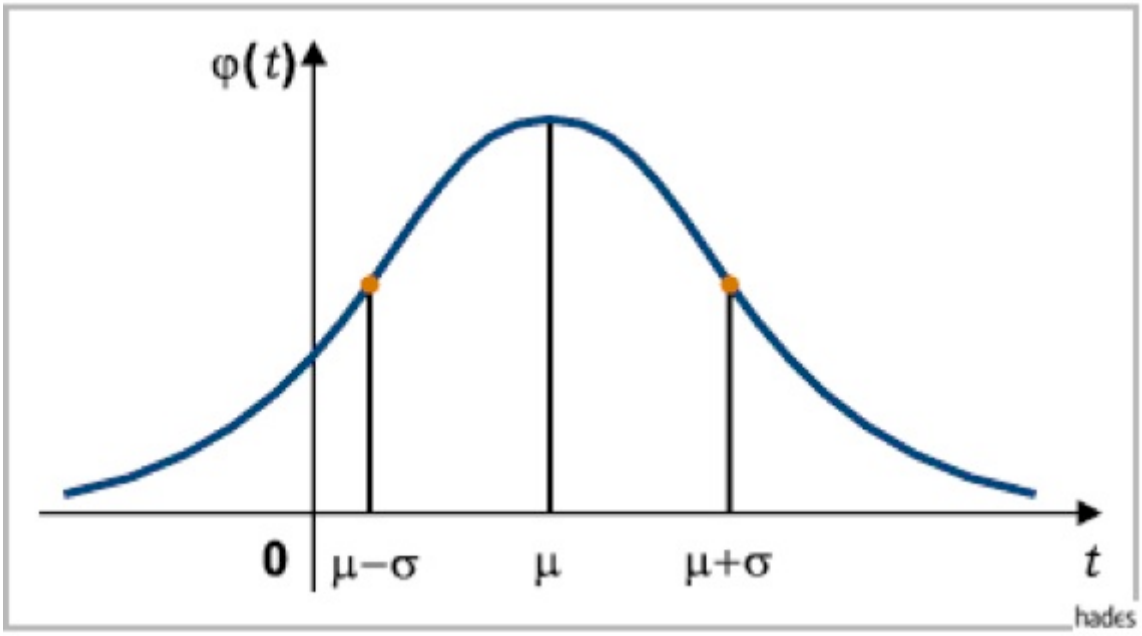


Figure 3.4: Normal distribution

We have seen that the averages – mean, median and mode – are very important for describing data. While all three measures can be very useful, they are not sufficient. If you think about the following questions, you will get an inkling for why this is the case. Grab a pen and a sheet of paper and sketch your answer these questions:

How do you think a distribution graph for communiting durations would look on a scale of 0 to 3 hours?

How do you think a distribution graph for the height of adult people would look on a scale of 0 to 3 meters?

> An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

In these bare-bones distribution graphs you see how starkly different distributions can look. Note in particular how commutes can run the gamut between a couple of minutes and mor than two hours, while the height of adults hardly ever goes below 1m and doesn't substantially go over 2m.

So, even if we were working with two samples that have a similar mean, median or mode, way have no yet of comparing how the data is strewn across its range. This means that comparing average values is at best a hint at statistical peculiarities, but no reliable tool or even a "proof" of any theory. Butler speaks for all statisticians when he cautions: "If we have demonstrated a difference in, say, the means or medians for two sets of data, can we not just say that one is indeed greater than the other, and leave it at that? This would, in fact, be most unwise, and would show a lack of understanding of the nature of statistical inference" (1985: 68-9). If you have calculated the averages for two data sets you want to compare, you will need further measures to begin making any substantive claims about their differences. This is where dispersion comes in.

Dispersion refers to how far the individual data points are spread over their scale. The obvious and simplest candidate measure for identifying dispersion is the range. The range effectively refers to the highest and the lowest values you have in your data, so all you have to establish the range is look for the maximum and minimum in the data. There are some simple functions for this in R:

```
> max(scores)
[1] 255
> min(scores)
[1] 117
> range(scores)
[1] 117 255
```

You can also calculate the ratio between the two points:

```
> min(scores) / max(scores)
[1] 0.4588235
```

This ratio tells us that the lowest score is only 46% of the highest score.

While there is some information to be gleaned from the range, it is a rather bad measure of dispersion for two reasons. Firstly, it is extremely susceptible to outliers. A measuring mistakes, or one student handing in the empty sheet can distort what little information the range gives us. Secondly, the bigger the sample, the less reliable the range is. This is because the likelihood of having an outlier increases with the sample size. To avoid these problems, we need a measure of dispersion which takes all scores into account.

The first idea would be to calculate the sum of the differences of each observation from the mean. However, by construction the differences cancel each other out. The sum of all negative differences, i.e. differences calculated from values which are smaller than the mean, is exactly equivalent to the sum of all positive differences, i.e. those calculated from values larger than the mean.[2]

The better idea is to work with the sum of squared differences. Squaring renders each difference a positive value. Moreover, strong deviations are weighted more heavily. The sum of squared differences, divided by the number of observations (minus one) is what we call the variance:

$$Variance(x) = var(x) = (sd(x))^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

This formula defines the variance as the sum of the squared difference between each observation, from x_1 to x_n, and the mean. This sum is divided by the number of observations, n, minus one. We take n minus one rather than n because it has shown to work slightly better.[3]

It is common to use the standard deviation *s* instead of the variance *s2*. The standard deviation is defined as the square root of the variance:

$$Std.Deviation(x) = sd(x) = s(x) = \sqrt{var(x)} = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

The standard deviation tells us how much any observation deviates on average. In other words, the standard deviation shows how much we can expect an observation to deviate. Imagine for instance that we can add an additional observation to our sample. We can expect this additional observation to deviate from the mean by the standard deviation.

In a *perfect* normal distribution, we get the following behavior: 68% of all sample values lie within $\bar{x}1s$, which is to say within the mean plus or minus one standard deviation, and 95% of all values lie within the range of $\bar{x}2s$.

A brief comment on notation. In the preceding paragraph, we used *s* as a shorthand for the standard deviation of the data. This is how you will often see the standard deviation notated because we generally work with data samples taken from larger populations. In this chapter we already saw a histogram of the word lengths in six issues of the London Gazette in 1691. However, the Gazette has continuously been published since 1665, the current issue at the time of writing being number 62,760. The six issues we looked at are a vanishingly small sample from a much larger corpus. If we were to analyze word lengths in these six issues, we would denote the standard deviation with *s*. However, if we were to analyze word lengths in all issues of the Gazette, which is to say the whole "population", and calculate its standard deviation, we would use the Greek letter sigma: σ. So the conventional notations are:

s = standard deviation of sample

σ = standard deviation of population

Most of the time, you will not be able to access the data of a full population, and so you will get used to seeing the standard deviation notated as *s*.

Z-Score and Variation Coefficient

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=33

A value closely related to the standard deviation is the z-score. Say you want to find out how much a particular value

2. For a longer discussion of this approach, see page 41 in Woods et al. (1986).

3. If you are interested in reading more about this, look up "Bessel's correction".

x_i deviates from the mean $\bar{x}$. While you could subtract the mean from the value you are interested in, the deviation of individual observations from the mean usually calculated in relation to the standard deviation:

$$z = \frac{(x - \bar{x})}{s}$$

This is known as the z-score.

[H5P: Calculate the z-score for a value x = x-bar + s]

The z-score expresses by how many standard deviations an individual value deviates from the mean. As such, it is a normalization, abstracting away from the actual mean and value, allowing us to compare values from different (normally distributed) data.

[H5P: Sarah takes two exams. She scores 12 points in exam 1, and 80 points in exam 2. Exam 1 has a mean of 10 and standard deviation 1.5. Exam 2 has a mean of 60 and standard deviation 30. In which exam did Sarah achieve better results?]

Another important measure is the variation coefficient. The variation coefficient is an easy way to express the amount of dispersion of a distribution graph. You can calculate the variation coefficient by dividing the standard deviation *s* by the mean $\bar{x}$:

$$c_v = \frac{s}{\bar{x}}$$

Thus, the variation coefficient is a ratio which expresses how many percent of the mean the standard deviation is. If you calculate the variation coefficient using the formula above, the result is a fraction. If you want to present the variation coefficient in percentage terms, you can multiple the fraction by 100.

Exercises

In this section, you will see again how you can plot data and learn to calculate the variance and the standard deviation in R.

Variance in R

Begin again by assigning your data to a variable:

```
> x = scores
```

In a first step you can just plot the raw data using:

```
> plot(x)
```

Since the resulting graph is not easily interpretable, you may want to use a nested function to sort the data:

```
> plot(sort(x))
```

[insert image of plot here]

Once sorted you can see a typical pattern. The curve first ascends steeply and levels out a bit in the middle before rising steeply again. This is typical of the normal distribution. As you know, most of the values in normally distributed data cluster around the mean, which is what the more level, plateau-like values in the middle of the plot show. Further away from the mean there are fewer values, which means that the jumps between observations are relatively bigger. This is what you see in the steep increases on the other side of the mean.

Of course, now you can plot the scores in a histogram, like you saw above:

```
> hist(x)
```

To calculate the variance of these scores, you can use a predefined function:

```
> var(x)
```

Alternatively, you can also calculate the variance manually. For that, the first step is to subtract the mean from all the values. Try this:

```
> (x - mean(x))
```

It may be surprising that the output for this command is a list, or in terms of R a vector, with as many entries as there are scores in the input data. These values are the demeaned scores, which is to say the difference between each score and the mean. R automatically recognized that the input is a vector and that the operation has to be performed for each value in the vector.

If we, naively, implement the first idea we had for a measure of dispersion, we just calculate the sum of the demeaned values above:

```
> sum(x - mean(x))
```

As expected, the result is zero, or close enough to zero that the difference is negligible. It is not exactly zero because of a few rounding mistakes.

To calculate the actual variance, you can implement the formula we discussed above in R. You will recall, the formula look like this:

$$Variance(x) = var(x) = (sd(x))^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

This translates into R as:

```
> sum( (x - mean(x))^2 ) / (length(x)-1)
```

In words, we calculate the sum of the squared differences – the ^ symbol is used for the mathematical power in R – and then divide the result by the number of observations minus one.

Standard Deviation, z-score and variation coefficient in R

The standard deviation can be calculated as the square root of the variance:

```
> sqrt(var(x))
```

Alternatively, you can use the predefined function to calculate the standard deviation:

```
> sd(x)
```

Once you have the standard deviation, you can also calculate the z-score for individual observations. For instance, if you want to calculate the z-score for the first value, x_1, you can do the following:

```
> ((x[1]-mean(x)) / sd(x))
```

In the first part of the command, you select the first value in the vector and subtract the mean from it. You can see here that values in vectors can be accessed in the same way as individual values in data frames. Unlike the data frame, the vector is a one-dimensional object, so you do not need to use the comma to specify the dimension. If you know the position of the value you are interested in, you can simply write that in the square brackets.

Finally, you can calculate the variation coefficient using two predefined functions:

```
> sd(x) / mean(x)
```

The result is a small value here, so you can see that the standard deviation is only a small fraction of the mean. Or, to put it differently, the standard deviation is only about 15% of the mean, which shows us that there is relatively little dispersion in our data.

From Description to Inference

By comparing on the one hand the averages (mean, mode and median) and on the other hand the standard deviations of two sets of data, we already get a clearer picture. But still, because real distributions differ from the perfect normal distribution, they do not deliver reliable data. A very crude test of "how normal" a distribution is consists in calculating the mean, the median and the mode. Since they coincide in a perfect normal distribution, the amount of their differences gives a (very) rough idea of how closely a distribution is "normal". As Figure 4.4 shows, 95% of all values in a perfect normal distribution lie within $\bar{x} \pm 2s$. A value outside this interval (95% is a common confidence interval) can be said to be statistically "abnormal".

But often we want to compare entire populations instead of single values, particularly in variationist linguistics. From which point on are 2 samples different because the population differs, and when are the differences only due to chance? Since we often want to compare sets of data, and since most distributions are not perfectly normal, different tests are needed. They exist in fact, one of them is the chi-square test, which we will use in inferential statistics.

References:

Butler, Christopher. 1985. *Statistics in Linguistics*. Oxford: Blackwell.

Lazarton, Anne, Heidi Riggenbach and Anne Ediger. 1987. Forming a Discipline: Applied Linguists' Literacy in Research Methodology and Statistics. *TESOL Quarterly*, 21.2, 263–77.

Oakes, Michael P. 1998. *Statistics for Corpus Linguistics*. Edinburgh: EUP.

Woods, Anthony, Paul Fletcher and Arthur Hughes. 1986. *Statistics in Language Studies*. Cambridge: CUP.

Part II

Inferential Statistics

4

INTRODUCTION TO INFERENTIAL STATISTICS

In this part of the book, Inferential Statistics, we discuss how we can move from describing data to making plausible inferences from it. To this end we introduce the concept of the standard error and discuss two common significance tests. In this chapter, we doscuss the normal distribution and the standard error, which will provide the basis for the significance tests in the next two chapters.

Inferential Statistics

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

Think, for a moment, about variationist linguistics, where we want to show that two speaker groups, two genres, two dialects and so on differ in certain features. In the last chapter we discussed some measures which provide a solid basis for statistically describing data. However, when we want to isolate which features are actually used differently in different variatieties of English, mere description is no longer sufficient, since differences we observe could be due to chance.

In the last chapter we saw that we can determine whether an individual observation is statistically abnormal or not. With normally distributed data, we know what the chances are that a particular value deviates by so and so much from the mean. We saw in Figure 3.5 that if we add a new observation from the same population to our sample, we can expect it to fall into the mean ± 1 standard deviation interval with a likelihood of about 68%. The likelihood that the new observation deviates from the mean by maximally ± 2 standard deviations is about 95%. If we add a new data point, and our new addition is a value lying outside this range, we could say that it is statistically quite unexpected, or, in colloquial terms, "abnormal".

Inferential statistics take us to another level. In inferential statistics we do not want to describe the relationship of individual data points to a sample. Indeed, normally we don't have a single observation about which we want to find out whether it is really unusual. Instead, we usually want to compare two samples from two different populations or compare one sample to a linguistic model. In other words, we want to know what we can infer about whole populations from one or two samples. And so the question we are dealing with in inferential statistics is: when can we infer that a difference is "real", which is to say statistically significant?

What is the normal distribution?

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

Remember that we only need two measures to describe normally distributed data: the mean and the standard deviation.

The mean defines where the center of the distribution is, and the standard deviation measures the dispersion of the data. If you are not clear on how this works, you may want to revisit the last chapter.

How does this normal distribution arise? It is best to consider this through some experiments. For these experiments, let us toss coins. The coin toss offers the easiest random outcome that you can imagine: we either get a head or we get a tail. If we toss a fair coin, what are the chances?

Of course, the chance that we get a head is 50% and the chance that we get a tail is 50%. The conventional notation for probabilities states the outcome of interest in brackets and gives the probability as a fraction. So if we toss a coin once, we get the following probabilities:

$p(head) = 0.5$ and $p(tail) = 0.5$

How do the probabilities change if we toss the coin twice?

$p(head, head) = 0.5 * 0.5 = 0.25$, so there is a 25% chance of getting 2 heads.

$p(tail, tail) = 0.5 * 0.5 = 0.25$, here, there is also a 25% chance of getting 2 tails.

$p(head, tail) = 0.5 * 0.5 = 0.25$, again, there is a 25% chance of getting 1 head and 1 tail.

$p(tail, head) = 0.5 * 0.5 = 0.25$, so too there is a 25% chance of getting 1 tail and 1 head here.

Since we are interested only in the number of occurrences and not in the sequence, the order is irrelevant. Accordingly, we can reformulate the last two probabilities into a single one:

$$p(1 \text{ head and } 1 \text{ tail}) = p(\text{head, tail}) + p(\text{tail, head}) = 0.5$$

Now we have two outcomes that occur with a likelihood of 25% and one outcome with a 50% likelihood of occurring. Again we can reformulate the probabilities of two coin flips, and state the probable outcomes:

$$p(0 \text{ heads}) = 0.25$$
$$p(1 \text{ head}) = 0.5$$
$$p(2 \text{ heads}) = 0.25$$

We can represent these outcome probabilities in a simple histogram:

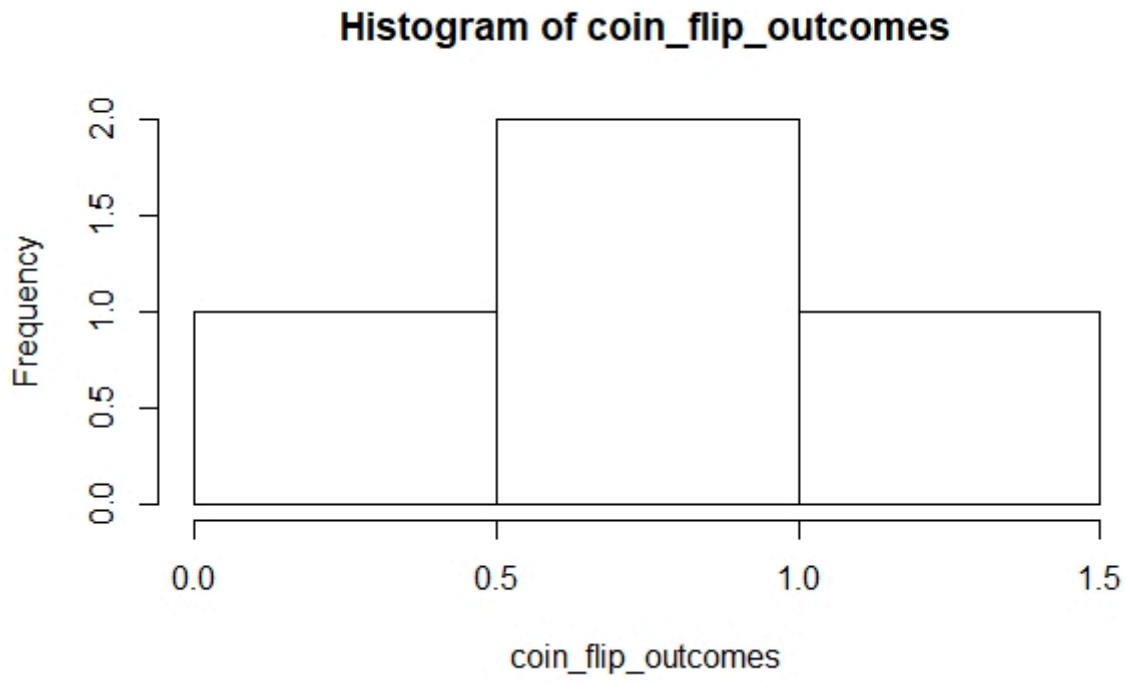


Figure 4.1: Outcome probabilities of two coin flips

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

If we toss the coin ten times, how many heads and tails will we get?

On average, we will get five of each, but the real outcomes vary a lot. Since it becomes tedious to flip coins so many times, let us use R to do the work.[1] We can use the `rbinom()` function to do this:

```
> rbinom(1,1,0.5)
[1] 1
```

`rbinom()` is a predefined function which generates a random binomial distribution. The name of the binomial distribution derives from the fact that there are only two possible outcomes. The probabilities of these two outcomes add up to one. It is one of the simplest statistical outcomes we can think of, and accordingly it is very easy to model.

Let's look at the three arguments. The last one is set to 0.5. This is the probability for the outcome. Since we are assuming that our coin is fair, we want a 50% chance of getting a head and a 50% chance of getting a tail.

By changing the first argument, we can tell R to flip a single coin several, let's say 10, times:

```
> rbinom(10,1,0.5)
 [1] 0 0 1 1 0 1 1 0 0 0
```

The second argument allows us to toss multiple coins. Consider the following command:

```
> rbinom(1,10,0.5)
[1] 9
```

We can think of this result as 9 of the 10 coins showing a head and one coin showing a tail. Of course, we can adjust the first and the second argument:

```
> rbinom(10,10,0.5)
 [1] 2 5 4 9 5 6 5 4 5 5
```

Now, on the basis of the second argument, R tosses 10 coins and sums the outcome, and on the basis of the first argument, R repeats this experiment 10 times. Since the list of numbers R generates as output of this command is not very intuitive to interpret, we should also visualize the results in a histogram. We can do this with the following sequence of commands:

```
> x = rbinom(10,10,0.5); x; hist(x)
 [1] 7 6 4 3 6 7 8 6 3 5
```

If you run this sequence of commands several times, you may notice that the histograms fail to resemble a normal distribution most of the time. It might be necessary to run the experiment more often. To do this, you can adjust the first argument:

```
> x = rbinom(100,10,0.5); x; hist(x)
> x = rbinom(1000,10,0.5); x; hist(x)
> x = rbinom(10000,10,0.5); x; hist(x)
```

In the slideshow of the histograms to these lines of code, you can see that the histogram looks more and more like a normal distribution, the more often you run the experiment:

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

1. Although we should not complain about flipping coins ten times, considering that two undergrads at Berkley undertook 40,000 coin tosses for an experiment (https://www.stat.berkeley.edu/~aldous/Real-World/coin_tosses.html).

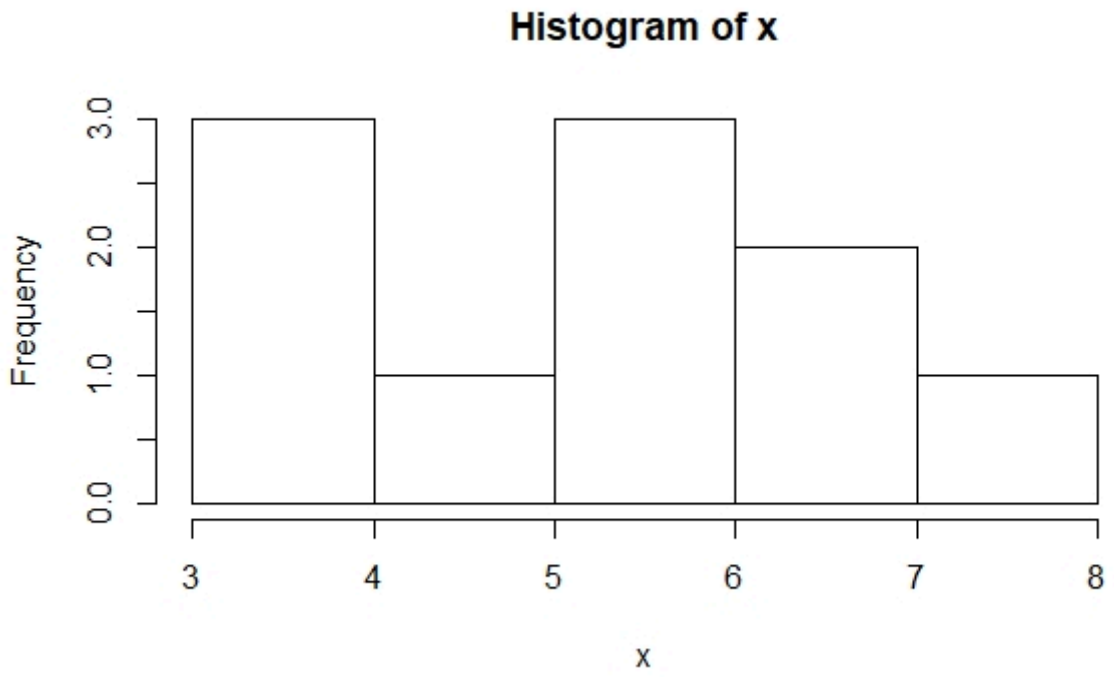


Figure 4.2: Histogram of 10 coins flipped 10 times

The following example from Johnson (2008: 11) shows that if we process these histograms a bit further, we can really come to see the normal distribution. First, let's assign a random normal distribution with 10,000 observations, a mean of 5 and a standard deviation of 2 to the variable *x*:

```
> x=rnorm(10000,5,2)
```

Then, we create a histogram with 100 intervals that shows values between 0 and 10. The logical argument `freq=F` plots the histogram as a density function, which means that the total area taken up by the histogram is one. And over this histogram we lay the curve of a normal distribution, `dnorm()`:

```
> hist(x,breaks=100,freq=F,xlim=c(0,10))
> plot(function(x)dnorm(x,mean=5,sd=2),0,10,add=T)
```

So far, we have only looked at symmetrical normal distributions with equal probabilities for both potential outcomes. However, many normal distributions are skewed, meaning that one of the outcomes has a higher likelihood of occuring. We can easily model this by changing the probability in the `rbinom()` function:

The missing element

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

We have seen that normal distribution is an intrinsic property of binomially distributed data. If you flip a fair coin often enough and summarize the outcomes, the results will necessarily be normally distributed. The essential point is that if

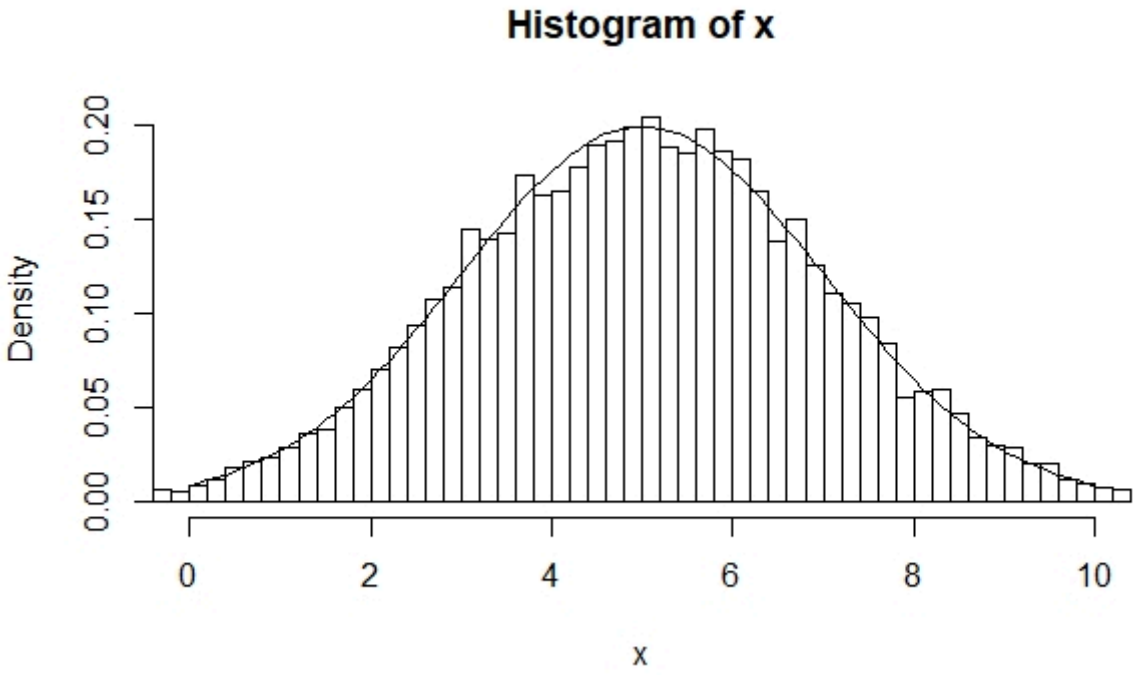


Figure 4.3: From histograms to the curve of a normal distribution

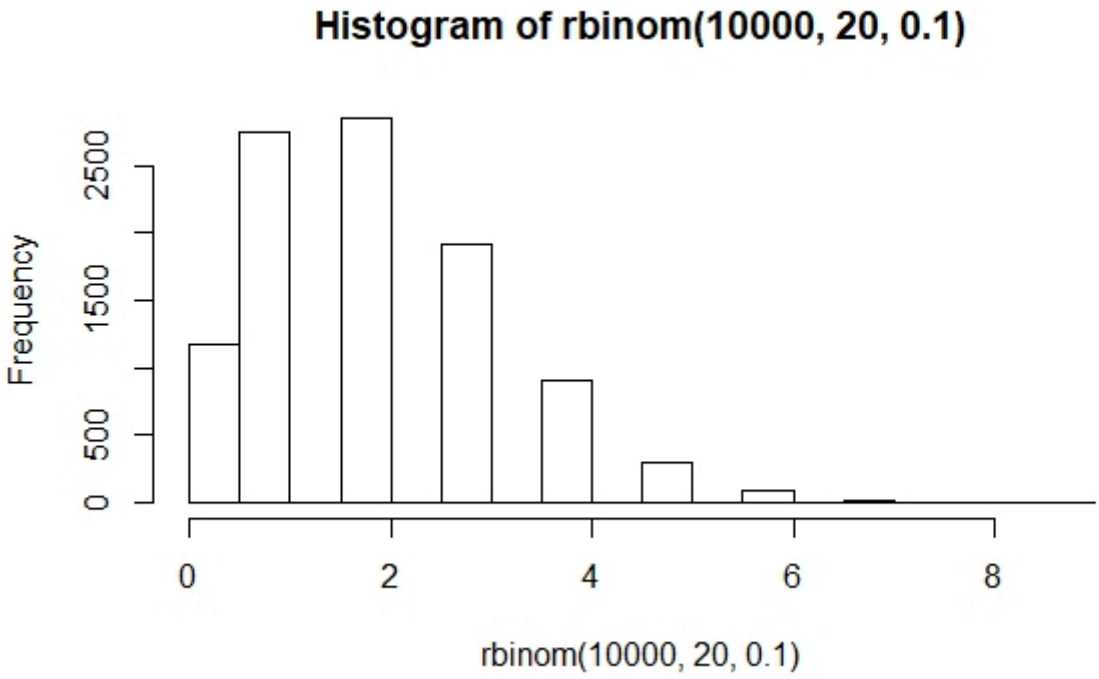


Figure 4.4: Histogram of a right-skewed normal distribution

we have normally distributed data, we can compare it to normal distributions derived from a model or from a different sample. The test that we will use for this comparison in the next chapter is the t-test.

As we have seen, normal distributions are fully defined by the mean and the standard deviation. For the t-test, which we will discuss in the next chapter, we need the mean and the standard deviation, as well as a measure for how reliable our sample is. If our samples are normally distributed, the only factor that restricts reliability is the size of our sample: as we have seen, with small samples, real outcomes vary a lot. The smaller the sample is, the more chance variation we expect. We discuss this chance variation, which is called the standard error, below.

If we have distributions that are not normal, different test for statistical significance are needed. They also exist, and we introduce one of them later on, in the chapter "Chi-square Test".

Take a look at the four decisively un-normal distributions below:

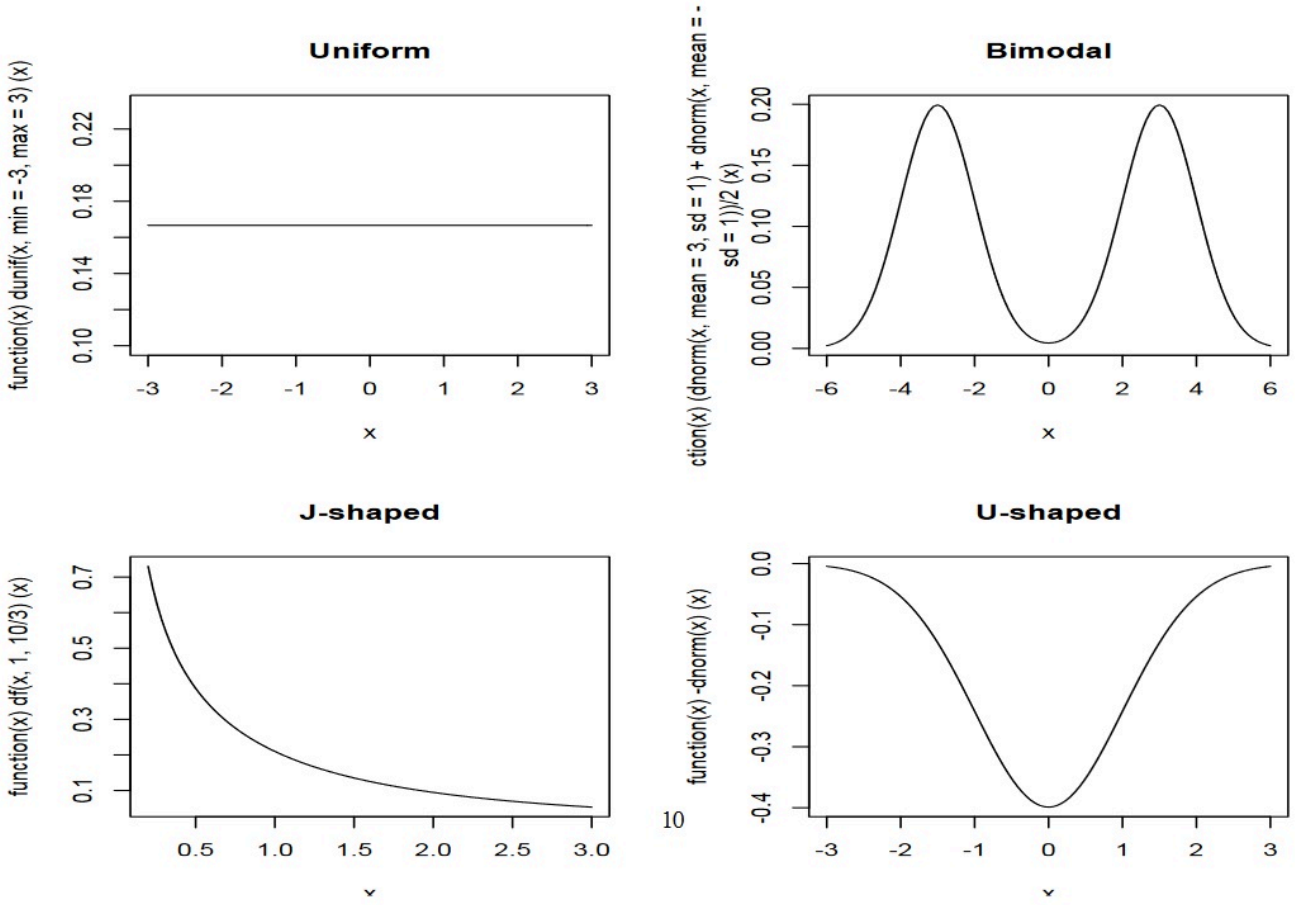


Figure 4.5: Uniform, Bimodal, J-shaped and U-shaped distributions

Now take a moment to think about how you expect the distribution of these measures to look. When you use the distributions above as inspiration, focus on the shapes, not the absolute values:

> An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

How normal is linguistics? How normally is the data distributed?

There is some dispute about how realistic the assumption of normal distribution is in linguistics. Taking a positive stance, Johnson claims that "the data we deal with [in linguistics] often falls in an approximately normal distribution" (16). Conversely, Oakes says that "in corpus analysis, much of the data is skewed" which is why for instance Chomsky criticized the corpus linguistics approach (1998: 4).

For our purposes it suffices to know that "skewness can be overcome using lognormal distributions" (Oakes 4), which means that rather than working with raw data we take the logarithms of our observations which often results in a normal distribution. Oakes also points to the central limit theorem which states that "when samples are repeatedly drawn from a population the means of the sample will be normally distributed around the population mean ... [no matter] whether

or not the distribution of the data in the population is itself normal" (Oakes 1998: 5f). In other words, the process of repeatedly taking samples generates a new, normally distributed set of data which is independent of the underlying population distribution. This is precisely what we did above when we simulated 10'000 times 10 coin flips with R (although in this example the underlying distribution is also normal).

Standard Error

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

As mentioned above, we need one last measure for chance variation before we can proceed to the first significance test. This measure is the standard error.

We have seen above that we get closer to the normal distribution the bigger our sample is. But the truth of the matter is that even a million coin tosses will not get us a *perfect* normal distribution. There is always a gap between outcomes and model (even if the coin is perfectly fair), between the sample and the (infinite) population, between "reality" and "fiction" (metaphorically speaking). We saw this earlier in Figure 4.3.

The gap between the observed data and the model is called the standard error, $\sigma_{\bar{x}}$, and it can be measured. We have seen that, on average, a single observationx_i deviates from the mean $\bar{x}$ by the standard deviation $s(x_i)$. The standard deviation expresses how much we can expect a token to deviate from the mean.

As the observation, so the sample: a sample deviates from the mean of the model, for instance an infinite population or a perfect normal distribution. The sample deviates from the model mean μ by the standard deviation of its mean $s(\bar{x})$. Now, the standard error expresses how much we can expect a sample to deviate from its perfect model. In other words, the standard error is a measure of how reliable our estimate $\bar{x}$ of the model mean μ is. This is why we can say that the standard error does for the sample with respect to the model what the standard deviation does for the observation with respect to the sample.[2]

The standard error is defined as follows:

$$\sigma_{\bar{x}} = \sqrt{\frac{s^2_{\bar{x}}}{n}} = \frac{s_{\bar{x}}}{\sqrt{n}}$$

To put the math into words, the standard error is the square root of the sample variance divided by the number of samples. Since we know that the variance is the squared standard deviation, we can reformulate this and say that the standard error is equal to the standard deviation divided by the square root of the number of observations.

Imagine the simplest case: you have only one observation. In that case, you can calculate the standard deviation and you know that n is 1. Since the square root of 1 is 1, you know that the standard error is equal to the sample's standard deviation.

If n equals 2, there is 50% chance that one observed value will be bigger than the model's mean, while the other will be smaller. If that is the case, then the deviations partly cancel out each other. If n=3 the chance that all observations are bigger or all are smaller than the model's mean is only 2 out of 8, so the mutual cancelling out of the deviations will become bigger and bigger when we increase n.

The standard error also makes sense intuitively: the more samples we have, the more observations we have. With more samples, the likelihood of our data representing the underlying population closely is higher. Accordingly, the standard error decreases by construction with more samples. This explains why the number of samples is in the denominator. Conversely, if the standard deviations in our samples are large, the likelihood of our data representing the underlying population closely is lower. Therefore, the standard error increases as the standard deviations increase. This explains the count.

The standard error in R

2. You will recall that we used σ in the last chapter to denote the standard deviation in the unknown, infinite population. Here we similarly use $\sigma_{\bar{x}}$ to refer to a measure of the unknown population.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

We can try to get an impression of the standard error in the following way. Use the following sequence of commands to generate a histogram of a binomial distribution:

```
> rbinom(100,10,0.5) -> x ; mean(x); hist(x)
[1] 5.12
```

This should look familiar to you by now, so we don't need to include a figure of the histogram here. Now run this line of code:

```
> rbinom(100,10,0.5) -> x ; mean(x); points(mean(x),-0.5)
[1] 5.11
```

Here it gets interesting: if you look at the space between the x-axis and the beginning of the histogram's bars, you will see that a small dot appeared. This is the mean of the freshly generated random binomial data. If you run this line multiple times – retrieve it using the up-arrow and just enter it several times – you will see that the dots all center around the mean of 5, but only very rarely hit it precisely.

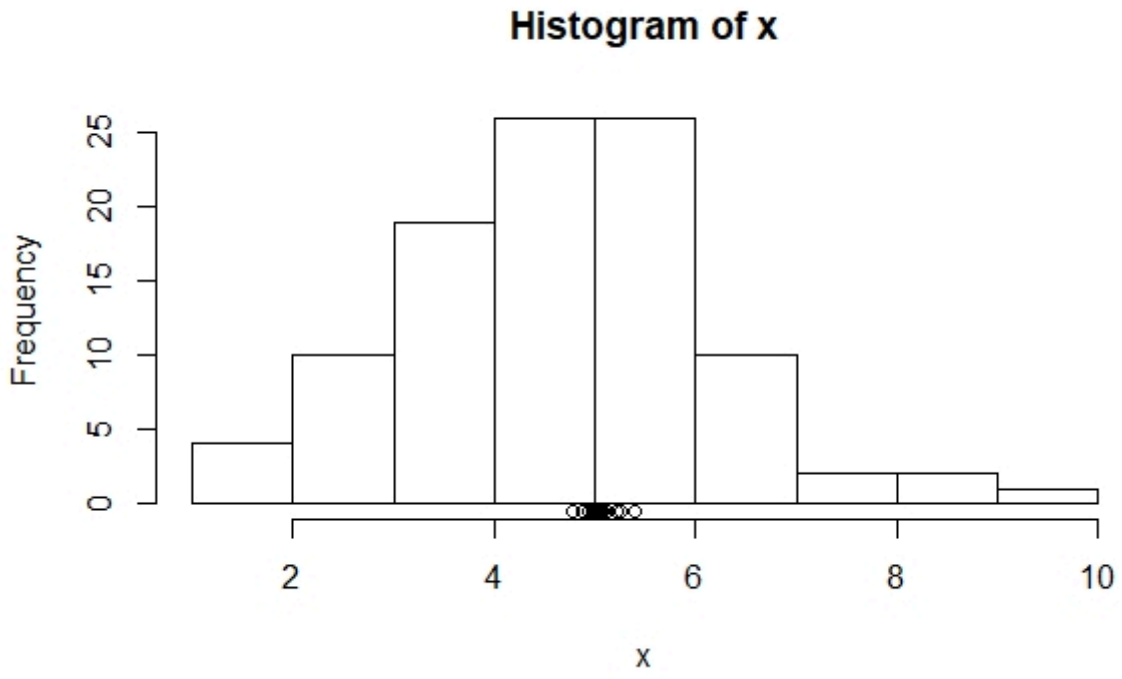


Figure 4.6: The sample means cluster around the model mean

What becomes visible here is how the sample means deviate much less from the underlying model mean of 5 than the individual values, which you can see in the bars of the histogram, do. This is because the standard error has the square root of the sample size in the denominator. The bigger the sample size becomes, the smaller the standard error is, and the more precise we can consider our sample to be.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=36

Think back to the beginning of the chapter. We said there that if we have normally distributed data and want to compare an individual observation to its sample, we can not say that the observation is really unusual if it lies within ± 2 standard deviations from the mean. However, if it lies outside this range, being either higher than the mean + 2 standard deviations or smaller than the mean – 2 standard deviations, we only have a roughly 5% probability that this value deviates so much from the mean by chance.

Now, we have seen that samples deviate in exactly the same way as tokens, but to a much lesser degree. Samples deviate by the standard error, while individual observations deviate by the standard deviation. This means that the same reasoning can also be applied to find out if a sample is significantly different from its model or not. And this precisely is what we are going to do with the t-test in the next chapter.

References:

Johnson, Keith. 2008. *Quantitative Methods in Linguistics*. Malden, MA: Blackwell.
Oakes, Michael P. 1998. *Statistics for Corpus Linguistics*. Edinburgh: EUP.

5

THE T-TEST

In this chapter, we discuss the t-test, with which we can test whether differences between sets of normally distributed data are statistically significant.

Prerequisite: standard error

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

We recommend that you refresh your understanding of the standard error in order to follow this introduction to the t-test. You can either watch the video above or revisit the last chapter to do so. If you are confident in your understanding of the standard error, you can plough on and warm up on these two questions:

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

Recall that with normally distributed data we can tell with fantastic precision what the likelihood is of an additional observation deviating from the mean by ± 1 standard deviation. Similarly, we could tell what the likelihood is of an additional sample deviating from the model mean by ± 1 standard error. You will not be surprised that the condition "with normally distributed data" is the reason for what follows. Since we never have perfectly normal data, or know the perfect model, the best we can do is assume that the data we observe in our samples correspond to the model. This is where the t-distribution comes in.

The t-distribution

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

To get from the normal distribution to the t-test, we need two elements. The first is the z-score, which we discussed in the Basics of Descriptive Statistics. You will remember that the z-score is a normalization which abstracts away from

individual values. Specifically, the *z*-score allows you to calculate the distance of an individual value to the sample mean, in relation to the standard deviation:

$$z_{x_i} = \frac{x_i - \bar{x}}{s_x}$$

A similar abstraction can be made to calculate the *z*-score of an individual value in relation to the underlying model, the infinite population. Instead of the standard deviation *s* and the sample mean $\bar{x}$ we use the model mean μ and the standard error σ:

$$z_{x_i} = \frac{x_i - \mu}{\sigma_x}$$

Now, we can also calculate the *z*-score for a whole sample with respect to the underlying model. In this case, we substitute the individual x value with the sample mean:

$$z_{\bar{x}} = \frac{\bar{x} - \mu}{\sigma_{\bar{x}}}$$

While these steps[1] are all important, they are also clearly only a theoretical exercise since, in most cases, we have no perfect model. Instead, we usually have two samples which we want to compare. The upshot of this is, of course, that we have no way of knowing μ or $\sigma_{\bar{x}}$. What we do know, however, is that we can calculate the standard error $\sigma_{\bar{x}}$. Remember that in the last chapter we defined as follows:

$$\sigma_{\bar{x}} = \sqrt{\frac{s_x^2}{n}} = \frac{s_x}{\sqrt{n}}$$

Because we usually know the standard deviation s and the sample size n, we can estimate the standard error above. Substituting the estimate for the standard error into the definition of $z_{\bar{x}}$, we arrive at this formulation:

$$z_{\bar{x}} = \frac{\bar{x} - \mu}{\sigma_{\bar{x}}} = \frac{\bar{x} - \mu}{\frac{s_x}{\sqrt{n}}} = \frac{(\bar{x} - \mu)\sqrt{n}}{s_x} = t$$

This is known as the t-distribution.

T-distribution and normal distribution

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

The t-distribution is closely related to the normal distribution, as you can see below.

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

What becomes visible when you juxtapose these two curves is that the peak of the t-distribution is a little lower than that of the normal distribution, while at the same time the tails on either side are bit fatter. We discuss why this is below, but first we'll show you how to plot these two distributions in R. The command we need for this is called `plot()`:

```
> plot(function(x)dnorm(x), -3, 3, col=1)
```

With this command, we plot a normal distribution between -3 and 3, obviously with the $\mu = 0$. The color parameter 1 corresponds to black.

Now we can add a red t-distribution to the graph with the following command:

```
> plot(function(x)dt(x,1,0), -3, 3, add=T, col=2)
```

The logical parameter `add` is set to be TRUE, and accordingly this red new curve is added to the previous figure. The second argument in `dt()`, set to one here, corresponds to the degrees of freedom (more on which later). See what happens when we increase the degrees of freedom, adding the green and blue curves, respectively:

1. For a more detailed explanation of them, see chapters 6 and 7 in Woods, Fletcher, Hughes

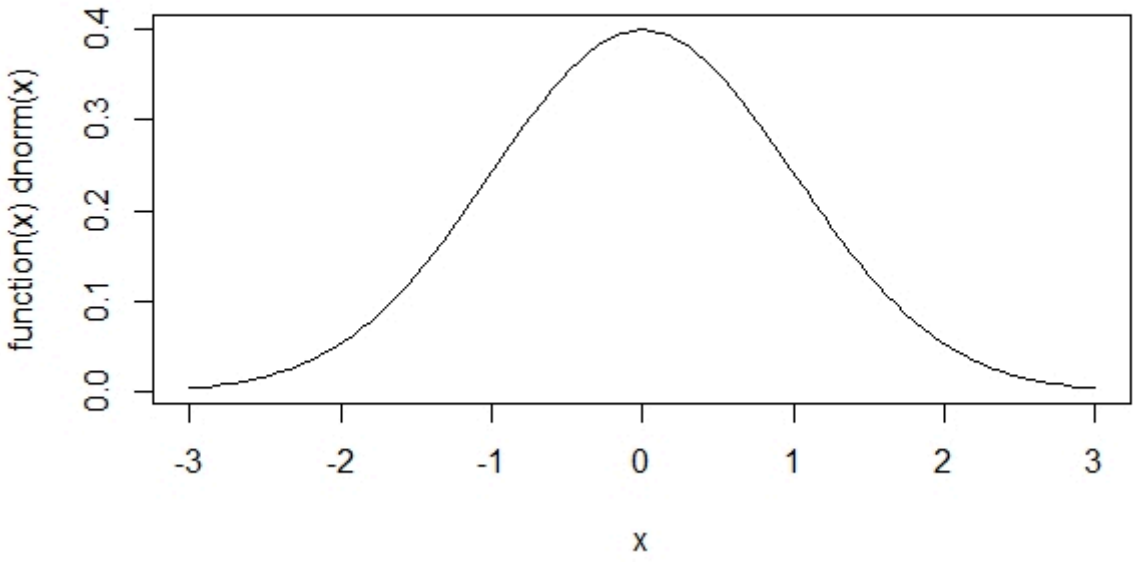


Figure 5.1: A normal distribution with mean = 0

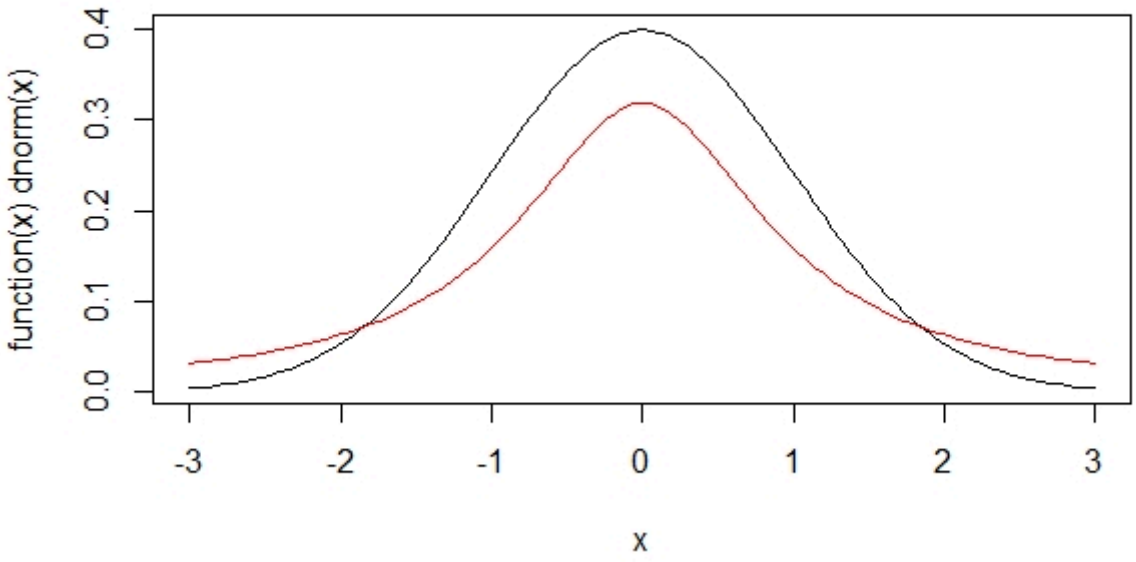


Figure 5.2: The normal distribution and a t-distribution with 1 degree of freedom

```
> plot(function(x)dt(x,10,0), -3,3, add=T, col=3)
> plot(function(x)dt(x,50,0), -3,3, add=T, col=4)
```

We see here that the more degrees of freedom we have, the more the t-distribution comes to resemble the normal

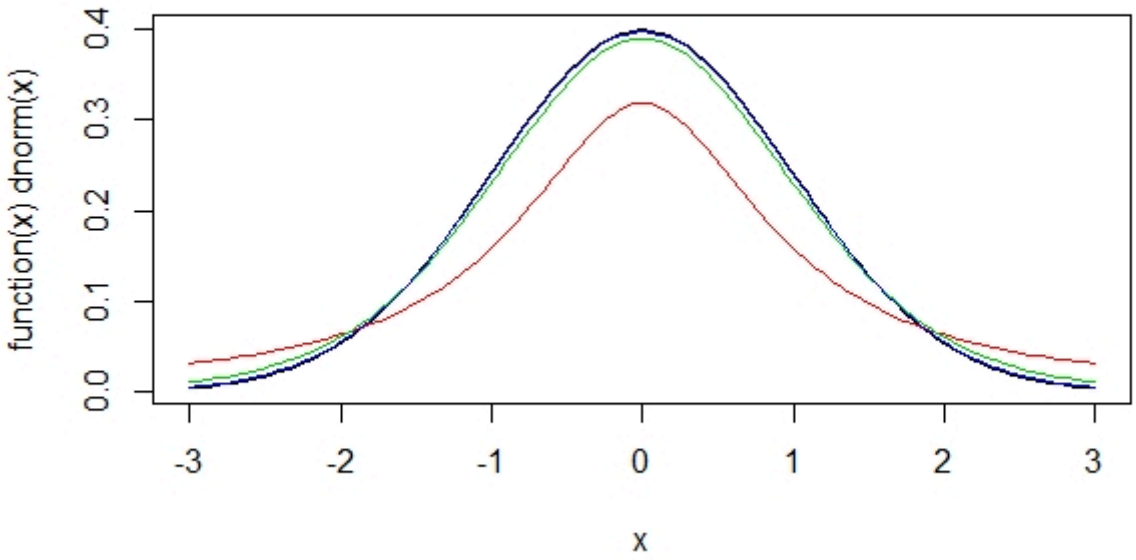


Figure 5.3: The normal distribution and three t-distributions with different degrees of freedom

distribution. The degrees of freedom are determined by the sample size. We know that the more observations we have in our samples, the smaller the difference between the samples and the model get. Indeed, if we have a t-distributions with many degrees of freedom, many starting at around 50, the t-distribution is virtually indistinguishable from a normal distribution with the same mean and standard deviation.

So why do we need both the normal and the t-distribution? This becomes clear when you recall how probabilities relate to the normal distribution. In the normal distribution, we know that 68% of all values range around the mean ± 1 one standard deviation and that 95% of all values lie in a range of the mean ± 2 standard deviations. We saw above that a t-distribution with one degree of freedom looks like a squashed normal distribution. This means that we have different probabilities for values to fall into a certain range. We saw that the t-distribution has a lower peak and fatter tails than the normal distribution. As a consequence, with the t-distribution we expect a higher probability for an individual observation to fall outside of the mean ± 2 standard deviations range.

So the t-distribution accounts for the fact that we are working with sample data which may deviate from the model. One could say that the t-distribution is a more careful version of the normal distribution. With the degrees of freedom parameter, the t-distribuion errs on the side of caution when we expect large differences between the sample and the model, i.e. when we are working with small samples.

The concept of the degrees of freedom is technically quite advanced and complex to understand. For our purposes, it suffices to understand Baayen's gloss of the concept: "Informally, degrees of freedom can be understood as a measure of how much precision an estimate has" (2008: 63).

So, here we are: If two largish, normally distributed samples have $z_{\bar{x}} > 2_2$" title="Rendered by QuickLaTeX.com" height="15" width="48" style="vertical-align: -3px;">, they can be said to be 'really' different.

From the t-distribution to the t-test

In order to understand the essence of the t-test, you need to understand a couple of points. The standard error is an intrinsic characteristic of all data, and it refers to the fact that every sample deviates from its population. Indeed, samples deviate from the population mean the same way observations deviate from the sample mean, but to a lesser degree. In particular, in a normal distribution:

- 68% of all tokens deviate by max. ±1 standard deviation, and 95% of the tokens deviate by ±2 standard deviations
- 68% of all samples deviate by max. ±1 standard error, and 95% of the tokens deviate by ±2 standard errors

Therefore, if a sample deviates by more than ±2 standard errors, chances that it is really different are above 95%.

As we saw above, the t-distribution is simply a conservative, i.e. more careful, version of the normal distribution. In being more conservative, it accounts for the fact that we are working with sample data, rather than an infinite population or a model. Practically, the t-distribution is more conservative than the normal distribution because we deduct the standard error. Because significance tests are usually performed on sample data, we work with the t-distribution and not the normal distribution.

Since the t-test is quite simple to understand in practice, we discuss it as we demonstrate how it works in R.

The t-test on generated data

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

There is a predefined function in R for the t-test, helpfully called `t.test()`. Let's start by taking a look at the help file:

```
> ?t.test
```

From the help file, we know that we can use the `t.test()` function to perform "one and two sample t-tests on vectors of data". At the moment, the important information for us is that the t.test is to be performed on vectors. There is, of course, a lot of other information in the documentation, but there is nothing to worry about if you only understand a small part. What we discuss in this chapter should be sufficient to perform t-tests, and by the time you require more detailed measures, you will probably have transcended the introductory level.

Now, let's try the t-test out on some fake data. To do that, create two objects containing random normal distributions with the same parameters:

```
> x = rnorm(100,5,2)
> y = rnorm(100,5,2)
```

Both objects, x and y, contain 100 observations of normally distributed data with a mean of 5 and a standard deviation of 2. Since R generates a random set of values, albeit with the same distribution, x and y are not identical, as you can see by looking at the two objects:

```
> x ; y
```

We refrained from printing the output here, since this list of numbers is not really interpretable to the human brain. Just note that there is a decent amount of variation in these randomly generated, normally distributed datasets. Now we want to apply the t-test to x and y:

```
> t.test(x,y)
```

```
> x<-rnorm(100,5,2)
> y<-rnorm(100,5,2)
> t.test(x,y)

        Welch Two Sample t-test

data:  x and y
t = 0.3319, df = 195.41, p-value = 0.7403
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.4910110  0.6897148
sample estimates:
mean of x mean of y
 5.109615  5.010263
```

Figure 5.4: Output to the t.test() in R

In the output, what interests us is the p-value. In the image above, we have a p-value of 0.74 which can be interpreted as a probability of 74% that the true difference in means is 0. In other words, we have a probability of 74% that the differences we observe are due to chance. Of course, the result of the t-test depends on the random data R generated for you, so you will see different numbers. In the video tutorial, we get p-values of 0.111, 0.146 and 0.125, all of which are surprisingly low. Surprising, that is, because we know that both of our samples have the same parameters.

Most of the time, we are interested in finding low p-values because we usually want to show that the observed difference between two samples is due to a real difference, rather than due to chance. In those cases, even our "low" p-values like 0.111 are too high, since they admit for a more than 10% probability that the differences we observe are due to chance. To provoke a lower p-value, let us generate a new sample with different parameters:

```
> y=rnorm(100,4,2)
```

Now we can run a t-test on the new y and the old x to compare two samples with different properties:

```
> t.test(x,y)
```

```
> t.test(x,y)

        Welch Two Sample t-test

data:  x and y
t = 5.2469, df = 190.07, p-value = 4.107e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.8956397 1.9747203
sample estimates:
mean of x mean of y
 5.063561  3.628381
```

Figure 5.5: T-test with two different samples

Most likely, the p-value you see in the output will be very small. In the image above we get a p-value of $4.107*10^{-7}$ In the video, we get a p-value of $7.685*10^{-6}$. This means that the probability of observed differences being due to chance is almost zero. When you run the t-test, you will get a slightly different value to these, but most likely it will also be a value very close to zero. If you were to get, say, a p-value of 0.06, the p-value might have you think "oh, ok, maybe these two samples aren't that different" when in fact you know they are they are: we know that because we created them above. This is, of course, very unlikely to happen, but it is possible.

In the line below the p-value, we see the sentence "alternative hypothesis: true difference in means is not equal to 0". This line has to do with the way statistical testing has developed conceptually: when we are performing significance tests, we are testing something we refer to as the null hypothesis. Usually, the null hypothesis (Ho) is something along the lines of:
Ho: The true difference in means is equal to zero

Yet, usually, we are interested in finding true differences that are not equal to zero. On the basis of Ho, we can formulate an alternative hypothesis (HA) more consistent with this interest:
HA: The true difference in means is not equal to zero

What you need to wrap your head around is that statistical significance testing does not allow us accept a hypothesis, we can only reject a hypothesis. This means that if we want to make a claim about a true difference between two samples, we need to reject the Ho, which states that there is no difference between the two means. And the way to reject the Ho is to show that the means we observe are unlikely to have arisen by chance. This is the information contained in the p-value. The p-value tells us what the likelihood is of observing the means that we have *under the assumption that the null hypothesis is true*. If we see that there is only, say, a likelihood of 3% that the differences we observe are the product of chance, we can reject the null hypothesis, toss away the idea that there is no difference between our samples, and continue working with the alternative hypothesis.

In the example with the really different *x* and *y* variables, we got a p-value of $4.107*10^{-7}$. This means that the chance that we would observe the same results if there were difference between the two variables is smaller than 0.1% by many order of magnitudes. In other words, we could draw a thousand samples from these two infinite populations, one with a mean of 4 and the other with a mean of 5, and never get a p-value high enough to not reject the Ho.

So you would probably expect that these two samples, so significantly different from each other, would look very different. Let's plot them and find out. We have already discussed distribution graphs at some length, so we can jump right into it. First, we can plot *x* as a histogram:

```
> hist(x,breaks=10,xlim=c(0,10),col=7)
```

Then we can add *y* to the same graph. To render *y* it transparent, we use this odd color code:

```
> hist(y,breaks=10,xlim=c(0,10),add=T,col="#0000ff66")
```

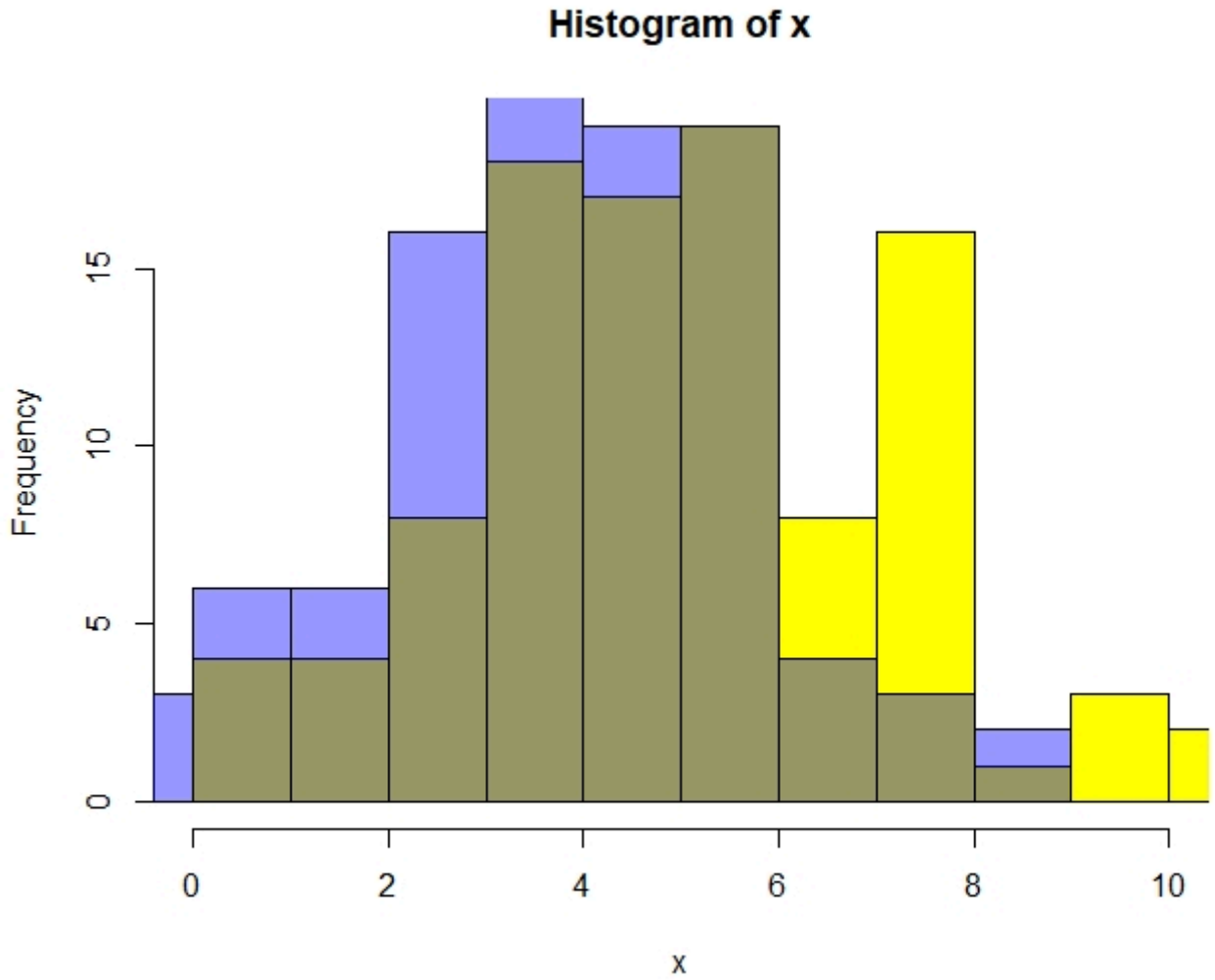


Figure 5.6: Our two samples, same-same but different

Although the histogram makes the difference between x and y visible, it is extremely hard for humans to judge whether a difference is significant or not. This is exactly why we need significance tests like the t-test.

The T-test on real data: The distribution of modals in six varieties of English

Let us look at an application of the t-test with real data. In this section we will not only apply the t-test, but discuss its properties, requirements and limitations.

An interactive or media element has been excluded from this version of the text. You can view it online here:
https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

Let us begin with an example adapted from Nelson (2003: 29). Suppose you want to study the distribution of modal verbs in different varieties of English. Take a look at table 5.1 below. It contains the five most frequent modal verbs in six varieties of English (British, New Zealand, East African, Indian, Jamaican and Hong Kong English, to be precise). The numbers are given as normalized frequencies per 1,000 words.

Table 5.1: Modal verbs in six varieties of English

	BE	NZE	EAfrE	IndE	JamE	HKE	mean	standard deviation
must	0.5	0.57	0.81	0.69	0.51	0.36	0.57	0.16
should	0.93	0.94	1.71	1.22	1.05	1.82	...	0.39
ought to	0.1	0.01	0.04	0.06	0.15	0.1	...	0.18
need to	0.27	0.34	0.32	0.02	0.83	0.33	...	0.26
have (got) to	1.65	1.85	1.39	1.35	1.56	2.36	...	0.37

An interactive or media element has been excluded from this version of the text. You can view it online here:
https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

Few authors publish their entire data, which can make research difficult sometimes. For this exercise, we have partly reconstructed the data from the International Corpus of English (Greenbaum 1996). The International Corpus of English, or ICE as it is also known, is a collection of corpora of national or regional varieties of English with a common corpus design and a common scheme for grammatical annotation. Each ICE corpus consists of one million words of spoken and written English produced after 1989, and comprises 500 texts (300 spoken and 200 written) of approximately 2,000 words each. Download the two files containing the frequency of modals (automatically _MD-tagged tokens, since you ask how we got these numbers) per document in the ICE India and the ICE GB corpora:

modals_iceindia.txt and modals_icegb.txt

Load the files into R:

```
> indiamod <- scan(file.choose(),sep="\r")  ## select modals_iceindia.txt
Read 97 items
> gbmod <- scan(file.choose(),sep="\r")     ## select modals_icegb.txt
Read 97 items
```

And then run a t-test:

The p-value here is 0.099, which means that there is a probability of almost 10% that difference we observe between the use of modals in Indian and British English is due to chance. You could, perhaps, choose a confidence value of 10% and argue that the difference between the two samples is, in fact, significant. You could do this because there is no universal rule which tells us which significance level to choose. In this absence of rules, there are only conventions. In linguistics, a 5% confidence value is the norm.[2] The confidence value is the p-value at which you are comfortable rejecting the Ho. So,

2. The confidence values depend heavily on the discipline. In linguistics 95% is a common cutoff point, which means that we are happy if we can establish that there is only a 5% (p-value = 0.05) chance that the differences we observe are random. In contrast, for genome-wide association studies in biology there is evidence to suggest that significance levels should be as high as 99.99999821%, which means that you should aim for a confidence value of 0.00000179% ($1.79*10^{-7}$) (Qu et al.).

```
> t.test(gbmod,indiamod)

        Welch Two Sample t-test

data:  gbmod and indiamod
t = 1.6599, df = 190.38, p-value = 0.09859
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.6000122  6.9711462
sample estimates:
mean of x mean of y
 25.98969  22.80412
```

Figure 5.7: Results of the t-test for the difference between modals in ICE India and ICE GB

we have to conclude that the differences between the use of modal verbs in the Indian and British ICE are not statistically significant.

Again, let's take a look at the data:

```
> hist(gbmod)
> hist(indiamod)
```

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=42

What we see are two histograms which are fairly typical of real outcomes of normal distributions. The Indian data is perhaps somewhat skewed to the right, but a slightly skewed normal distribution still counts as a normal distribution.

Let's zoom in on one specific modal: should. Download the two files containing the counts of should (should_MD tokens) per article in ICE India and ICE GB:

should_iceindia.txt and should_icegb.txt

Load them into R and run a t-test:

```
> indiashould <- scan(file.choose(),sep="\r") ## select should_iceindia.txt
Read 97 items
> gbshould <- scan(file.choose(),sep="\r")    ## select should_iceindia.txt
Read 97 items
```

```
> t.test(indiashould, gbshould)

        Welch Two Sample t-test

data:  indiashould and gbshould
t = 2.0276, df = 156.88, p-value = 0.04429
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.02398187 1.83168824
sample estimates:
mean of x mean of y
 2.505155  1.577320
```

Figure 5.8: Difference between the use of should in the ICE India and ICE GB

This time we get a p-value of 0.044. If we go with the conventional 5% significance value, we could say that there is a significant difference between the frequency of should in the Indian and British varieties of English.

However, if we plot the data, we might have second thoughts about making that claim:

```
> hist(gbshould)
> hist(indiashould)
```


These two histograms do not look like normal distributions. Not even like skewed normal distributions which grow strongly on one side and have a long tail on the other. Instead, what we get are many cases where there is no, or perhaps a single, should, and then some documents in which there are surprisingly many shoulds (the maximum number of *should*s lies at 15 and 23, for British and Indian English, respectively).

This means that one of the assumptions of the t-test, which assumes that the data is normally distributed, is violated. Thus, although we have what looks like a significant difference, we have to be extremely careful with how we interpret the results.

We are in a situation here where "we hoped for clarity, but just get new questions". There is good reason to think that this is not as depressing as it may seem. Without statistical tests, there would be no room for nuance. Things would be either blindingly obvious or impossible to distinguish. With the t-test, however, we have a basis for making the claim that the difference in the use of should in Indian and British English is most likely significant, as long as we are transparent about the significance level and our doubts regarding the distribution of the data. Moreover, we know that this is a marginal case and that it might be worthwhile to investigate the question further. Thus, the t-test, and statistical significance testing in general, can be useful even when the results do not offer any obvious conclusions.

Exercises

Outlook

We now understand the big picture, painted with a big brush. We can now successively turn to some of the gory details.

Everything we did, concerning the t-test, we did in R. But if you look at the definition of the t-distribution in the exercise above, you see that it is possible to calculate the t-test manually given that you know the mean, the standard deviation and the sample size for two samples. So far, we focused strongly on the p-value, and completely neglected the t-value, which is also always included in the R output of the t-test. When we compared *should* in Indian and British English, our output included a value of t = 2.0276. This is also the result you get if you enter the means, standard deviations and sample size into the the formula for Welch's two-sample t-test:

$$t = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

When not working with a software like R, we could calculate the t-test manually, and would then have to look up the t-score for our specific degrees of freedom and the significance levels. These are found online, and most statistics books include one in the appendix. It's great to know that software does all of this for us, but it is really helpful to understand what the software is actually doing, and knowing that you could do it yourself.

If you continue using significance tests, or also just read quantitative research, you will become familiar with different versions of the t-test. Some of the most frequent ones are the one- and two-tailed tests, which are sometimes called one- and two-sided tests. Woods, Fletcher and Hughes state that the two-tailed test "is so called because values sufficiently out in either tail of the histogram of the test statistic will be taken as support for H1. ... one-tailed test, by contrast, is so called because values sufficiently far out only in one tail ... will be taken as support for H1" (122). In our *should* example

above, we had the null hypothesis that the two variants of English use *should* with the same frequency, in the hopes of rejecting it in favor of the alternative hypothesis (that there is a significant difference between the frequency with which the two variants use *should*). Because we were agnostic about which variant uses *should* more frequently, we used a two-sided test. If, however, we had the conviction that *should* occurs more frequently in Indian English than in British English, we could have conducted a one-sided test to identify whether IE *should* > BE *should*. For our purpose, the two-sided test often suffices, but if you are interested in reading up on the difference between one- and two-tailed t-tests we recommend reading Woods, Fletcher and Hughes (1986), pages 122 to 126.

We will discuss more applications of the t-test in later chapters, but you should now have a solid understanding of what we are doing when we are using the t-test. In the next chapter we discuss the chi-square test, which we can use on data which is not normally distributed.

References:

Greenbaum, Sidney. (ed.) 1996. Comparing English Worldwide: The International Corpus of English. Oxford: Oxford University Press.

Nelson, Gerald. 2003. "Modals of Obligation and Necessity in Varieties of English". In Pam Peters, editor, *From Local to Global English*. Dictionary Research Centre, MacquarieUniversity, Sydney, pages 25–32.

Qu, Hui Qi, Matthew Tien and Constantin Polychronakos. 2012. "Statistical significance in genetic association studies". *Clin Invest Med*, 33 (5): 266 – 270.

Woods, Anthony, Paul Fletcher and Arthur Hughes. 1986. *Statistics in Language Studies*. Cambridge: CUP.

6

THE CHI-SQUARE TEST

In this chapter, we discuss the chi-square test. The affordance of the chi-square test is that it allows us to evaluate data of which we know that it is not normally distributed.

The chi-square test

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=44

In the last chapter, we introduced the t-test and saw that it relies crucially on the assumption that the data in our samples is normally distributed. Often, however, our data is not normally distributed. For these cases, we can use different significance tests that don't assume a normal distribution.

Perhaps the most versatile of these is the chi-square test. In contrast to the t-test, which requires the mean, the standard deviation and the sample size, the chi-square test works with the differences between a set of observed values (O) and expected values (E). The expected values either come from a model or from a reference speaker group. Frequently, you will calculate the expected values on the basis of the observed values, thereby generating your own model. We discuss how this works in detail below.

To calculate the chi-square value, expressed by the Greek letter χ, the differences between observed and expected values are summed up. When we discussed the summation of differences in the context of the standard deviation, we saw that if we just sum the differences, positives and negatives cancel each other out. With the standard deviation, we work around this impasse by squaring the differences, and we can use the same approach to calculate the chi-square value. So, we square the difference between each observation and its corresponding expected value. Since we are not interested in the absolute differences, we normalize the squared differences before summing them, which gives us the following formula for the chi-square value:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

As you can see, we normalize by E, since the expected values are perhaps more balanced than the observed values. This is already most of what we need for the chi-square test. Once we have the chi-square value, we can look up the probability for our results and see whether they are significantly different than what would expect if our data were just a bunch of random observations. That's already the whole theory.

In the next sections, we discuss how the chi-square test is performed in practice.

Contingency tests

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=44

A frequently used version of the Chi-square test is the contingency test, in which the expected values are the random distribution of the observed values. To illustrate what this means, let's consider the following example which is based on Mukherjee (2009: 86ff). Here, we use data from the Lancaster-Oslo-Bergen (LOB) and the Freiburg-LOB (FLOB) corpora. The LOB contains texts from 1961 and the FLOB contains texts from 1991, and both corpora contain 500 texts of about 2,000 words each.

Let's use the chi-square test to establish whether the usage of *prevent NP from doing something* (form 1) and *prevent NP doing something* (form 2) changed in British English between the 1960s and the 1990s. If we look at the table below, it seems that in the 1960s form 1 is predominant, while in the 1990s both forms are equally likely. But is this difference statistically significant?

Table 6.1: Forms 1 and 2 observed

	form 1	form 2	row total
LOB 1960s	34	7	41
FLOB 1990s	24	24	48
column total	58	31	89

Before evaluting this, let's formulate the hypotheses. The hypotheses for the chi-square test function similarly to those for the t-test, which means that we want to phrase a null hypothesis which we can reject in order to continue working with the alternative hypothesis. So, we can say:
H0: there is no difference in the usage of these forms over time.
HA: there is a difference in the usage of these forms over time.

Now, we could just enter our data into R and run a chi-square test on it, and we are going to do so below. But since working through this test manually enables a better understanding for how the test functions, we are going to do this first.

Performing the contingency test manually entails six steps:

1. For every O calculate E
2. Calculate the deviance of E from O
3. Sum the values
4. Calculate the degrees of freedom
5. Decide on a significance level
6. Check/calculate the critical values

Let's walk through this step by step.

Step 1: For every O calculate E

The expected frequencies are defined as "the frequencies that we would expect simply by chance (if the independent variable had no relationship to the distribution)" (Hatch and Farhady 1982: 166). Let's look at an example before going into the LOB/FLOB data. If we look at foreign students enrolling for three different faculties at an American university, if there were 1,032 foreign students and three faculties (Sciences, Humanities, Arts) and the area of research were of no importance to students, we would expect that all faculties would get an equal number of foreign students, namely $\frac{1032}{3} = 344$. Our table would look like this:

Table 6.2: Expected foreign students

	Sciences	Humanities	Arts	Row total
Foreign students	344	344	344	1032

However, because we have access to the enrollment data, we know how the numbers actually look. And the enrollment rates you observe look like this:

Table 6.3: Observed foreign students

	Sciences	Humanities	Arts	Row total
Foreign students	543	437	52	1032

You can see that there is a substantial difference between our model, where all students are equally likely to enroll in any of the three faculties, and the observed data, which shows that some faculties clearly appeal to more students than others.

In our example with *prevent NP*, if the null hypothesis were true, we could assume that the two samples of British English from different periods are "a single sample from a single population" (Woods et al. 1986: 141). In other words, we assume that the number of occurrences for each of the two linguistic variants is independent of the time period. To calculate the expected values, then, we need the following formula:

$$E_{i,j} = \frac{(row\ i\ total)*(column\ j\ total)}{grand\ total}$$

For each cell, in row i and column j, we multiply the row total with the column total and divide the result by the grand total, which corresponds to the sum of all observations. Knowing this, you should be able to calculate the expected values for the observations in table 6.1. Write the expected values down in a table.

Step 2: Calculate the deviance of E from O

Remember that to calculate the χ^2 value, we will require the normalized differences: $\frac{(O_i-E_i)^2}{E_i}$. Use this formula to calculate the normalized squared difference. Again, write them down in a table.

Step 3: Sum the values

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i-E_i)^2}{E_i} = 2 + 1.7 + 3.73 + 3.18 = 10.62$$

Step 4: Calculate the degrees of freedom

We touched on the difficulty of degrees of freedom in the last chapter, but fret not: it is easy to calculate them for the contingency test. Butler states that the number of degrees of freedom "is determined by the fact that if we know all but one value in a row or column, and also the row and column total, the remaining cell can be worked out. Therefore degrees of freedom = (no. of rows-1)*(no. of columns-1)" (1985: 121). In our example, this means: df = (2-1)*(2-1) = 1.

Step 5: Decide on a significance level

We have to do this whenever we perform a significance test. Let us stick with the conventions of linguistics and aim for significance level of 95%. This means that we want to have a probability of 95% that the difference between the two sets of data from the two time periods is *not* a matter of chance. In terms of the confidence value we discussed in the last chapter, we would want a p-value ≤ 0.05.

Step 6: Check the critical value

The table above contains the critical values for the χ^2 value we calculated. How do we figure out the correct critical value? Well, we know that we have 1 degree of freedom, so we look at the first row, and since we decided on a 95% significance level, we look at the column under 0.05. Then we check whether the χ^2 we calculated in step three is bigger than the critical value:

$\chi^2 = 10.62 > 3.84_{3.84}$" title="Rendered by QuickLaTeX.com" height="19" width="138" style="vertical-align: -4px;">

Since the calculated χ^2 value is bigger than the critical value, we can reject the null hypothesis. In fact, the critical value at the 0.01 level is still smaller than our χ^2 value, wherefore we can conclude that the difference between the two sets of data is highly significant at the 1% level. In other words, there is a likelihood of 99% that the difference we observed in the usage of the two forms is not a matter of chance. This means that we can reject the Ho and assume that HA is true: there is a difference in the usage of the two forms over time.

	Significance level					
df	0.20	0.10	0.05	0.025	0.01	0.001
1	1.64	2.71	3.84	5.02	6.64	10.83
2	3.22	4.61	5.99	7.38	9.21	13.82
3	4.64	6.25	7.82	9.35	11.34	16.27
4	5.99	7.78	9.49	11.14	13.28	18.47
5	7.29	9.24	11.07	12.83	15.09	20.52
6	8.56	10.64	12.59	14.45	16.81	22.46
7	9.80	12.02	14.07	16.01	18.48	24.32
8	11.03	13.36	15.51	17.53	20.09	26.12
9	12.24	14.68	16.92	19.02	21.67	27.88
10	13.44	15.99	18.31	20.48	23.21	29.59

Figure 6.1: Table of critical values of chi-square (from Butler 1985: 176)

The reasoning behind this conclusion should become clear when you look at how the chi-square value is defined:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

The χ^2 corresponds to the total deviation of the observed and the expected values, and the difference between the two is too high for the Ho to be true.

Exercise: contingency test in Excel

Before moving to a higher abstraction, which is the chi-square test in R, try solving the contingency test in Excel. Use the formula discussed above to calculate the expected values, and then us the Excel function CHITEST. You can copy Table 6.1 into Excel to get started. In the video above, Gerold walks you through the steps, but we recommend you try it yourself first and check out the solution in the slides below:

As you can see, Excel automates one step for you: you no longer have to look up the critical value. Instead, the output of the Excel function is the probability of Ho being true. This means that you come to conclusion we had above simply by interpreting the Excel output correctly.

Chi-square distribution

In chapter 4 we saw how the normal distribution naturally arises. You probably were not really surprised when you saw the bell shape of the normal distribution (we weren't). Like the normal distribution, the chi-square distribution arises naturally, but its shape is far from intuitive.

We'll show you what we mean by this while we discuss how to generate a chi-square distributions in R. First, we define four normally distributed variables, each with mean of 0, a standard deviation of 2 and 10,000 observations:

```
> x=rnorm(10000,0,2)
> y=rnorm(10000,0,2)
> z=rnorm(10000,0,2)
> w=rnorm(10000,0,2)
```

If you visualize these as histograms, you will just see a normal distribution:

```
> hist(x, breaks=100, freq=F)
```

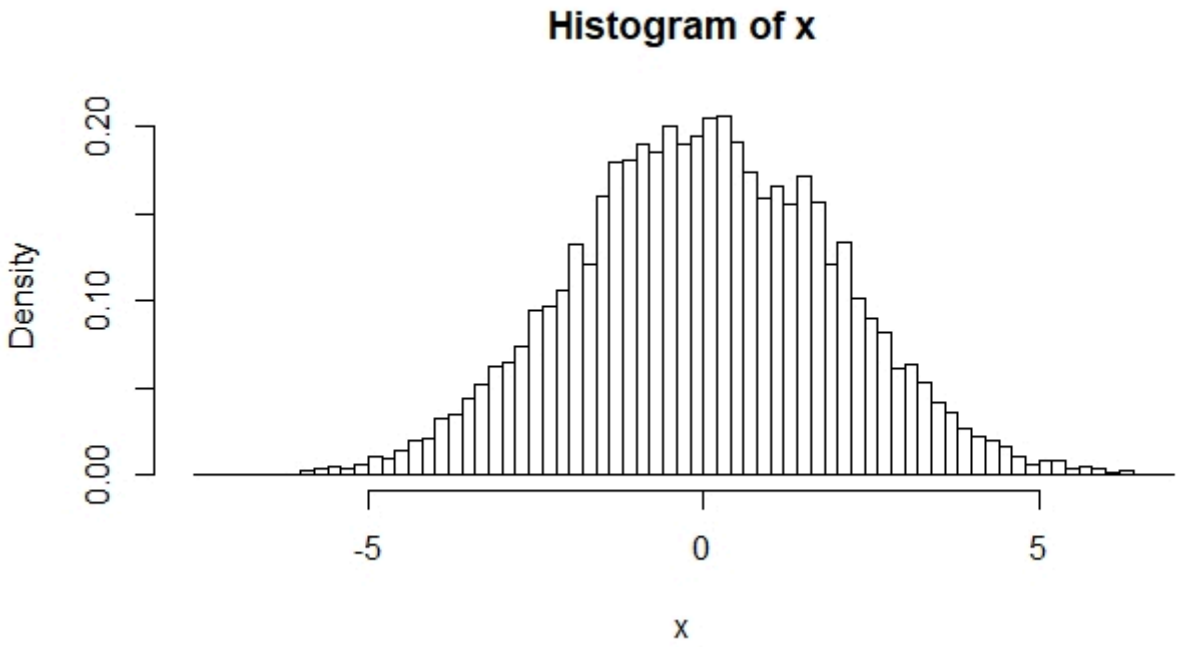


Figure 6.2: Histogram of x

This changes drastically if you square *x* and plot it again:

```
> x2=x^2
> hist(x2, breaks=100, freq=F)
```

We use the variables from above to generate sums of independent, squared and normally distributed random variables:

```
> xy2 = (x^2+y^2) # sum of 2 squared normal distributions
> xyz2 = (x^2+y^2+z^2) # sum of 3 squared normal distributions
> xyzw2 = (x^2+y^2+z^2+w^2)    # sum of 4 squared normal distributions
```

Now we can use these sums to plot the different chi-square distributions with different degrees of freedom:

```
> hist(x2,breaks=100,xlim=c(0,40),ylim=c(0,0.2),freq=F)
> hist(xy2,breaks=100,add=T,freq=F,col="#0000ff66") # df = 2
> hist(xyz2,breaks=100,add=T,freq=F,col="#ff000066") # df = 3
> hist(xyzw2,breaks=100,add=T,freq=F,col="#00ff0066") # df = 4
```

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=44

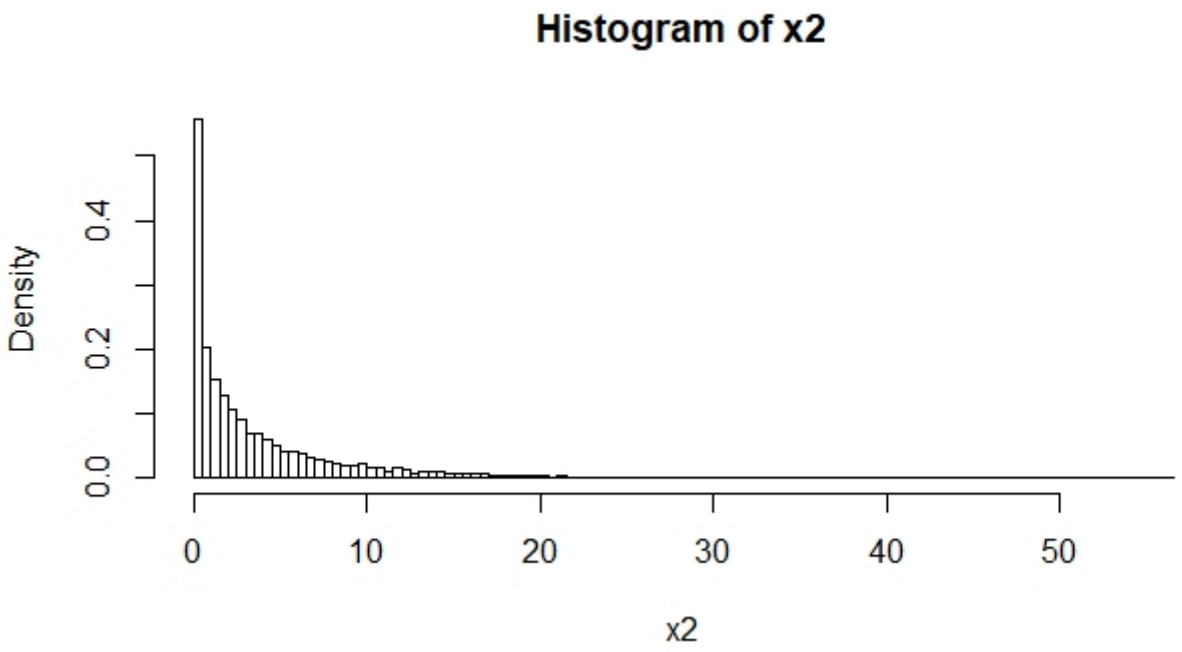


Figure 6.3: The variable x squared

You can see that the chi-distribution changes drastically with each change in the degrees of freedom. This is why it is essential to know how many degrees of freedom we have when running a chi-square test.

As with the t-distribution, we require the chi-square distribution to derive the critival value. In the example above, we calculated $\chi^2 = 10.62$ with df=1. Now, we visualized a chi-square distribution with 1 degree of freedom and saw that most of the values are very close to one. In fact, we looked up the critical value for the 0.05 significance level which told us that 95% of all values in a chi-square distribution are smaller than 3.84. So, the chi-square distribution functions as a benchmark for our expectations, just as the t-distribution did in the previous chapter.

Chi-square test in R

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=44

A note on the video: the links to the web calculators Gerold refers to are located at the end of the chapter.

We mentioned earlier that using R for a chi-square test is more abstract from the contingency test than using Excel is. The simple reason for that is that R does not require us to calculate the expected values. This is best illustrated with an example.

Let's replicate the *prevent NP (from) doing* data in R. We can do this easily by creating two vectors which contain the data. Assign the data from the 1960s to the variable `lob` and the data from the 1990s to the variable `flob`:

```
> lob=c(34,7) ; flob=c(24,24)
```

In the next step, we need to create a table from these two vectors. We can do this by assigning the output of the `data.frame()` command to a new variable:

```
> lobflobtable <- data.frame(lob,flob)
```

`data.frame()` takes the data variables as arguments and forms a table from them.

Now, we can simply use the inbuilt `chisq.test()` function on our table which contains the observed values. The expected values are calculated automatically and are not displayed. When you enter the chisq.test command, the output looks like this:

```
> chisq.test(lobflobtable)

        Pearson's Chi-squared test with Yates' continuity correction

data:  lobflobtable
X-squared = 9.1607, df = 1, p-value = 0.002473
```

You can see that R calculates χ^2 value and the degrees of freedom and gives us the p-value. With a p-value = 0.002473, the R output looks slightly different than our Excel results above. The reason for this is that R automatically includes the so-called Yates' continuity correction, which aims at alleviating an error introduced by the assumption that the discrete probabilities of frequencies can be approximated by a continuous distribution.

Pitfalls of chi-square: conditions of usage

As with every other significance test, the chi-square test has very clear conditions of usage that we have to consider in order to attain reliable results. There are three general conditions which should be considered when working with the chi-square test.

The first of these is that absolute frequencies have to be used for the chi-square test. If you convert absolute frequencies into relative or normalized frequencies, the chi-square value may appear highly significant when in fact it wouldn't be with absolute frequencies (Woods et al. 1986: 151). Thus, performing a chi-square test with normalized or relative frequencies is misleading, and should not be done.

The second important condition is that all observations have to be independent from each other (Gries 2008: 157). Woods et al. state that "whenever it is not clear that the observations are completely independent, if a chi-square value is calculated, it should be treated very skeptically and attention drawn to possible defects in the sampling method" (1986: 149).

The third condition is that "each observation must fall in one (and only one) category" (Hatch and Farhady 1982: 170). In other words, the categories have to be mutually exclusive.

Two further features of the chi-square test are concerned with specific cases. The first of these is the Yates' correction, which has to be applied "for one-way χ^2 and 2×2 χ^2 tables where the df is only 1" (Hatch and Farhady 1982: 170). The Yates correction factor is defined as follows: "0.5 is added to each value of O if it is less than E, and 0.5 is subtracted from each value of O it is more than E" (Oakes 1998: 25). Thus, the Yates' correction is applied in cases with a minimum degree of freedom. The second is that the chi-square test does not work with low expected frequencies. The frequencies are too low to work with when at least one of the expected frequencies lies below 5 (Mukherjee 2009: 88).

Then, of course, there is a more general problem with very high frequencies and the assumption of random generation of natural language: "For all but purely random populations, $\frac{(O-E)^2}{E}$ tends to increase with frequency. However, in natural language, words are not selected at random, and hence corpora are not randomly generated. If we increase the sample size, we ultimately reach the point where all null hypotheses would be rejected" (Oakes 1998: 28f, also see Mukherjee 2009: 89). This does not prevent linguists from using the chi-square test, and it does not generally invalidate the use of the chi-square test. But knowing this, you might want to consider using a different test if the chi-square test identifies things that look very similar as being different.[1]

Finally, you should be aware of what the chi-square test can and cannot be used for. The chi-square test "does not allow to make cause and effect claims" (Oakes 1998: 24). It will only "allow an estimation of whether the frequencies in a table differ significantly from each other" (Oakes 1998: 24). Thus, while the chi-square test is very useful to identify whether the patterns we observe in the data are random or not, it falls short of actually allowing us to make causal claims. In order to make those, we need some more advanced statistical methods, like the regressions we discuss in chapter 10.

Useful web links

Before we progress to the next chapter, we should point out that there are several online calculators which run the chi-square test for you. At the time of writing, these include:

- http://statpages.org/#CrossTabs

1. See, for instance, the 'chi by degrees of freedom' in Oakes 1998: 29

- http://www.physics.csbsju.edu/stats/contingency_NROW_NCOLUMN_form.html
- http://www.mirrorservice.org/sites/home.ubalt.edu/ntsbarsh/Business-stat/otherapplets/Catego.htm

Due to the nature of the web, links change very often, and we cannot guarantee that these links will work. If they do, however, they are a useful resource if you want to run a quick chi-square test without access to Excel or R.

References:

Butler, Christopher. 1985. *Statistics in Linguistics*. Oxford: Blackwell.

Gries, Stefan Th. 2008. *Statistik für Sprachwissenschaftler*. Göttingen: Vandenhoeck & Ruprecht.

Hatch, Evelyn Marcussen, and Hossein Farhady. 1982. *Research design and statistics for applied linguistics*. Rowley, Mass: Newbury House.

The LOB Corpus, original version. 1970–1978. Compiled by Geoffrey Leech, Lancaster University, Stig Johansson, University of Oslo (project leaders), and Knut Hofland, University of Bergen (head of computing).

Mukherjee, Joybrato. 2009. *Anglistische Korpuslinguistik – Eine Einführung*. Grundlagen der Anglistik und Amerikanistik 33. Berlin: Schmidt.

Oakes, Michael P. 1998. *Statistics for Corpus Linguistics Edinburgh*. Edinburgh: EUP.

Woods, Anthony, Paul Fletcher and Arthur Hughes. 1986. *Statistics in Language Studies*. Cambridge: CUP.

Part III

Language Models

7

1-GRAMS

In this part, Language Models, we discuss how statistics can inform the way we model language. To put it very simplistically, language consists of words and of the interaction between words. In this chapter we focus on language at the level of words, and show how we can use the methods discussed in the preceding section can help us to evaluate word frequencies.

The simplest language model: 1-gram

At its simplest, language is just words. Imagine language as an urn filled with slips of paper. On each paper there is one word, and together these words constitute the lexicon of the language. Say the urn contains N words. We draw M word tokens from the urn, replacing them every time. At the end of this exercise, we have a text which is M words long. If this is our language model, what are the assumptions behind it?

Basically, there are two assumptions in this model. The first of these is that *every draw is independent*. This means that the word you have just drawn has a probability of $\frac{1}{N}$ of occurring, regardless of what precedes it. Of course, this is not a realistic assumption. In actual language, the first word of a sentence narrows the number of possibilities for the second one, which in turn reduces the number of choices for the third word, and so on and so forth. We will discuss this assumption at greater length in the next chapter.

The second assumption is that *every word type is equally likely to be drawn*. Since each word has a $\frac{1}{N}$ probability of being drawn, it is equally likely that you will draw "the" and "Theremin". Again, you are quite right in thinking that, actually, not all words occur with the exact same frequency. What might be less obvious than the observation that different words are used with different frequencies is how frequent different words actually are. How likely, which is to say frequent, are frequent words? How frequent are rare words? How much more frequent are frequent words than rare words? One may also take a sociolinguistic perspective, and ask: who uses how many words? Actually, how many word types are there?

In what follows, you will see how the statistical methods from the last chapters can help to answer some of these questions.

Types, tokens, rare and frequent words

In order to explore the distribution of words in R, we need to load in a large corpus. For this chapter we use the ICE Great Britain. The file is structured as a long list with one word per line. In R terminology, we have a long vector of strings. Download the raw text file here:

[icegb_written.words]

Save it so you can find it again and load it into R:

```
> icegb <- scan(file.choose(),sep="\n", what="raw")
```

We can take a first look at our vector:

```
> icegb[1:20]
```

Now, we can restructure the corpus into a frequency table, using the `table()` command we introduced in earlier:

```
> ticegb = table(icegb)
```

If you look at the first few entries of the table, it doesn't look too informative yet. However, we can sort the table by descending frequency to see what the most frequent words are:

```
> sortticegb = sort(ticegb,decreasing=T)
```

And then take a look at the 100 most frequent words:

```
> sortticegb[1:100]
```

```
> icegb[1:20]
 [1] "Tom"         "Rodwell"     "With"
 [4] "these"       "profound"    "words"
 [7] "J."          "N."          "L."
[10] "Myres"       "begins"      "his"
[13] "book"        "&lsquo;"     "The"
[16] "English"     "Settlements" "&rsquo;"
[19] "&semi;"      "although"
>
```

7.1: The first twenty words in ICE GB

```
> sortticegb[1:100]
icegb
    the      of      to     and       a      in      is    that     for      be       I      as
  24614   13889   11821   10615    8611    7680    5348    4113    3944    3506    3158    2984
     it     was     The      on     are    with     you      by      's    have    from      at
   2907    2891    2818    2786    2631    2593    2385    2287    2269    2137    1972    1969
    not      or &lsquo; &rsquo;   which    this      an    will     but     had      he     has
   1947    1947    1905    1867    1845    1666    1492    1400    1357    1330    1252    1249
    his    been    were    they   would     can    more   their     one     all     her      It
   1225    1138    1116    1105    1060    1050     998     987     979     942     919     913
     so      if   there     she     may    your      In      we  &semi;    when   about     who
    840     828     824     814     794     786     753     753     750     737     734     721
   than      up     its     out    This &ldquo;    some    time    into      no   other    also
    713     706     694     672     671     670     668     668     664     664     660     643
&rdquo;    only     any   could    said    such      do       A    very    them    what      my
    639     639     623     568     565     557     543     531     531     525     514     513
  these    over      He  should     now      If      me    then    most     two   being   first
    501     495     494     490     485     482     481     460     459     449     434     429
 people     him     new     But
    428     425     424     416
> |
```

7.2: The 100 most frequent woords in ICE GB

Now, of course, we can begin to visualize the results, to make them more easily interpretable. For instance, we can create a set of histograms:

```
> hist(as.vector(sortticegb[1:100]),breaks=100)
> hist(as.vector(sortticegb[1:1000]),breaks=1000)
> hist(as.vector(sortticegb[1:10000]),breaks=10000)
```

[H5P slideshow: histograms]

[H5P: What type of distribution is this?]

We can also check for the number of different types in the ICE GB, using this sequence of commands:

```
> types = length(as.vector(sortticegb)) ; types
[1] 33816
```

[H5P: How would you find out the number of tokens in R?]

One obvious question that arises when looking at word frequencies is who uses how many words, or rather who uses how many different types? The measure which is used to evaluate vocabulary richness is known as the type per token ratio (TTR). The TTR is a helpful measure of lexical variety, and can for instance be used to monitor changes in children with vocabulary difficulties. It is calculated by dividing the number of types by the number of tokens in a passage of interest:

$$TTR = \frac{types}{tokens}$$

Using this formula, we can calculate the TTR of the ICE GB:

```
> types/tokens
[1] 0.07929615
```

Usually the TTR is very small, which is why it is often handier to invert the ratio and calculate the tokens per type:

$TTR = \frac{tokens}{types}$

In this formulation, the TTR tells us how often, on average, each type is used. When we refer to the TTR henceforth, we talk about the token per type ratio.

For the ICE GB, we would then calculate:

```
> TTR = tokens / types ; TTR
[1] 12.61095
```

This value indicates that, on average, each type occurs 12.6 times in the ICE GB .

As soon as we start to think more about this metric, we can see that it depends fundamentally on the sample size. The paragraph in which we introduce the TTR above – starting with "One obvious" and ending with "passage of interest" – consists of 82 tokens and 59 types. Accordingly, its TTR is $\frac{82}{59} = 1.39$. In this paragraph, each type occurs 1.39 times on average. Now, let us look at a very short sentence. For instance the preceding one. The TTR for "now let us look at a very short sentence" is 1, because every token is of a unique type. This means that when we compare different samples with each other, as we do below, we want to take samples of a comparable size.

Exercise: TTR

You can try this yourself now. You may wonder whether written language has a richer vocabulary than spoken language, as measured by TTR.

[H5P: What is the null hypothesis to formulate here?]

[H5P: Calculate the TTR for the first 10'000 words of the ICE GB written, which we used above.]

Now, download the ICE GB spoken from here:

[icegb_spoken.words]

Then, load it into R.

[H5P: Calculate the TTR for the first 10'000 words of the ICE GB spoken]

You can see that there is a pronounced difference between the spoken and written parts of the ICE GB corpus.

[H5P: Which significance test would you use to check whether the difference is statistically significant?]

With this basic principle, you have a solid basis for comparing the vocabulary richness of different genres, dialects or speakers. However, when it comes to producing fully fledged research, simply taking the first 10'000 words of two corpora is generally not advisable since the sequence of documents in a corpus can potentially skew the results. Instead, we recommend forming segments of equal length, for instance 2,000 words, calculating the TTR for each segment and then comparing the mean.

Zipf's law

Another interesting aspect of word frequencies in corpora is known as Zipf's law. Named after the Harvard linguistics professor George Kingsley Zipf (1902-1950), it explains the type of distribution that we get when counting the word tokens in a large corpus. Zipf's law states that in a sorted frequency list of words the measure $rank * frequency$ is nearly constant. In other words, there is an inversely proportional relationship between the rank of the type and the number of tokens this type contributes to the corpus.

We already sorted the written ICE GB corpus according to frequency above, so let's investigate how well Zipf's law holds up here. Remember that to view the contents of a table, we can use square brackets:

```
> sortticegb[1]
 the
 24614
```

To only retrieve the frequency in a table like this, we can use two square brackets:

```
> sortticegb[[1]]
[1] 24614
```

For the first ranked entry, this is of course already the value we need to test Zipf's law (because the frequency multiplied by 1 is equal to the frequency). For the remaining ranks, we can formulate a sequence of commands which calculates the rank*frequency:

```
> myrow = 3; freq = sortticegb[[myrow]] ; zipfconstant = freq * myrow ; zipfconstant
[1] 35463
```

By scrolling back up in the history in the R console, we can quickly generate a small sample of these values:

```
> myrow = 5; freq = sortticegb[[myrow]] ; zipfconstant = freq * myrow ; zipfconstant
```

```
[1] 43055
> myrow = 10; freq = sortticegb[[myrow]] ; zipfconstant = freq * myrow ; zipfconstant
[1] 35060
> myrow = 20; freq = sortticegb[[myrow]] ; zipfconstant = freq * myrow ; zipfconstant
[1] 45740
```

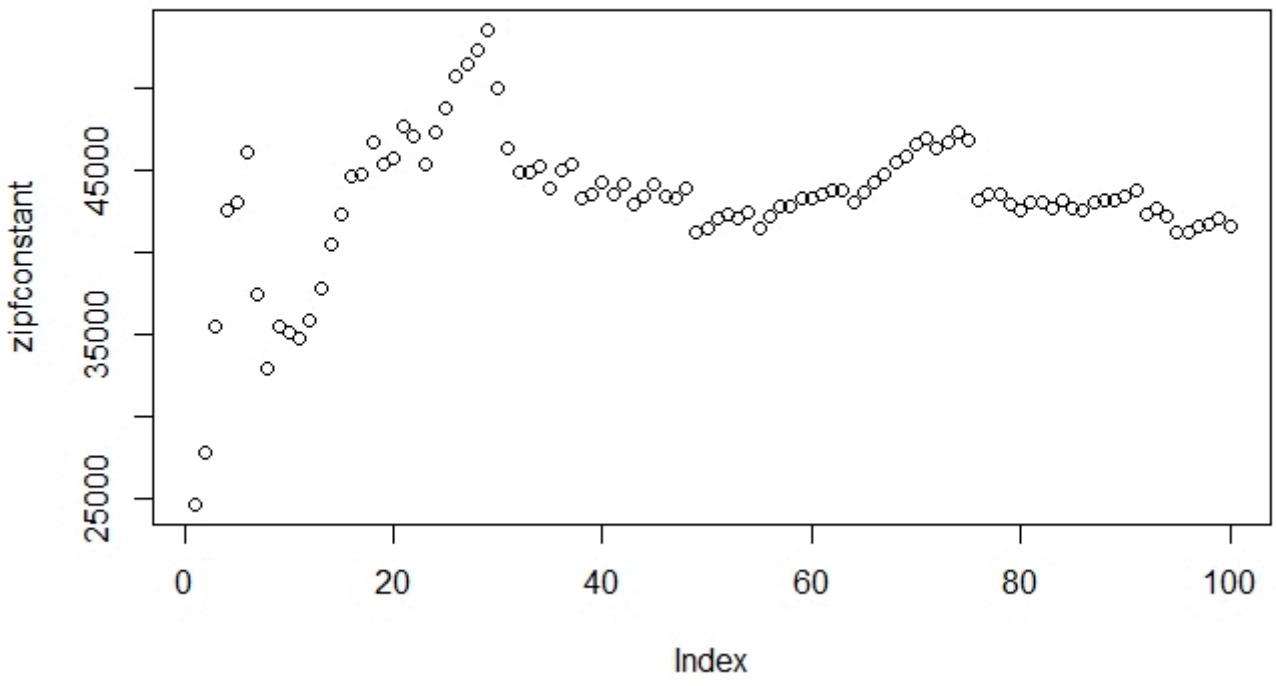
There is a lot of variance here, but five values are clearly not enough for us to draw any conclusions yet. Say we want to see whether Zipf's law holds up over the 30, or 100, most frequent words. Instead of assigning values from 1 to 30 (or 100) manually, we can tell R to go through each of them, using a loop. Remember that loops work exactly like the indexes in mathematical sums (if not, you might want to take a quick look at the Introduction to R again, where we discuss this for the first time).

In the loop, we first give the value 1 to the index variable i, then 2, and so on until we reach the last value of interest, 30 in this case. We save the results in the list variable titled zipfconstant, which we again index with i:

```
  > for (i in 1:30) { freq = sortticegb[[i]] ; zipfconstant[i] = freq * i}
> zipfconstant
 [1] 24614 27778 35463 42460 43055 46080 37436 32904 35496 35060 34738
     35808   37791 40474 42270 44576 44727 46674 45315 45740 47649 47014
 [23] 45356 47256 48675 50622 51435 52276 53505 49980
```

Now we have a vector, zipfconstant, which contains the Zipf contstant for the thirty most frequent words in the ICE GB. Even though these are only thirty values, it is rather hard to intuit anything about how constant they are, especially because the values are all very large. It might be easier to see patterns if we plot the values of a larger sample:

```
  > for (i in 1:100) { freq = sortticegb[[i]] ; zipfconstant[i] = freq * i}
> plot(zipfconstant)
```



7.3: The Zipf constant for the 100 most frequent words in ICE GB

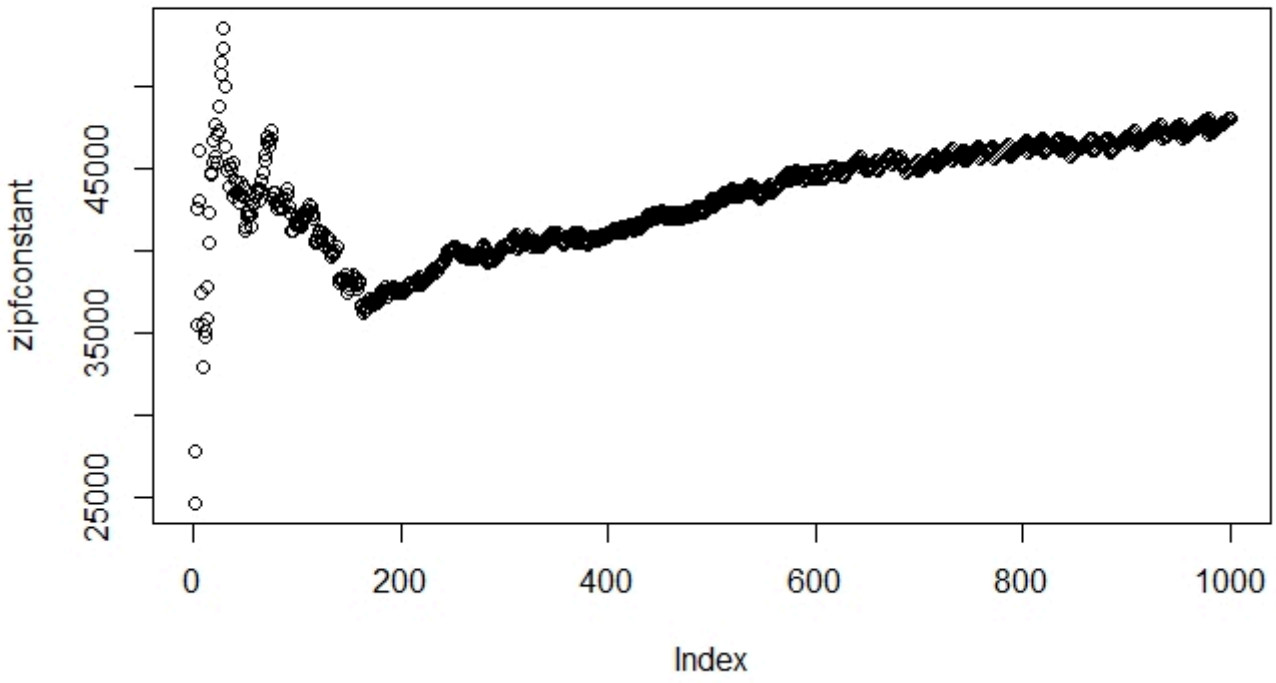
How can we interpret this? The plot shows us that there is a lot of variance in the Zipf constant of the highest-ranking words. This makes a lot sense, when we consider that the difference between these frequencies vary strongly. Let us calculate the Zipf constant for the five most frequent words:

- 24614*1 = 24614
- 13889*2 = 27778
- 11821*3 = 35463
- 10615*4 = 42460
- 8611*5 = 43055

There are large jumps in the frequencies, which cause a lot of variation in the Zipf constant. However, this is only true for the first thirty or so values. After those, we see a strong regularity, with the Zipf constant ascending steeply. This indicates that, for the highest ranking values, the increase in rank outweighs the decrease in frequency. Then, the constant begins to earn it's name.

Let's see what happens when we extend the analysis further:

```
  > for (i in 1:1000) { freq = sortticegb[[i]] ; zipfconstant[i] = freq * i}
> plot(zipfconstant)
```



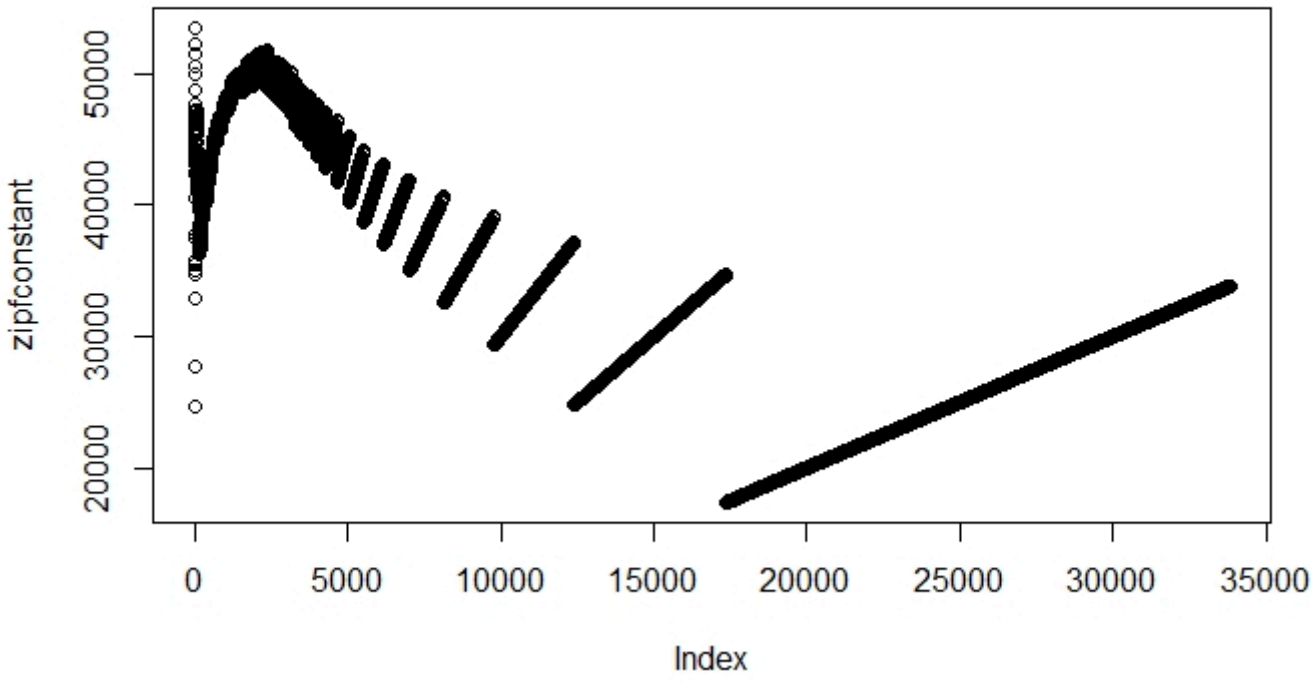
7.4: The Zipf constant for the 1000 most frequent words

When we look at the Zipf constant for the 1,000 most frequent words, we see a similar pattern. We retain the high variance in the highest ranking words, and then see a more stable pattern, a mild upwards trend, starting at around 180 on the index axis. However, the regular pattern we saw starting at around 30 in graph 7.3 looks a lot wilder here. What's going on here? Well, we compress the y-axis by a factor of ten, i.e. we fit ten times more observations on the y-axis, but the x-axis stays the same. So the variance, the vertical fluctuation, stays within the same range but is compressed horizontally. Accordingly, it looks as though the fluctuations are more intense when we plot more observations in a graph of the same size.

Let's see what happens when we plot the Zipf constant for the entire ICE GB corpus. To do this, we can let our loop run through the length of the corpus:

```
  > for (i in 1:length(sortticegb)) { freq = sortticegb[[i]] ; zipfconstant[i] = freq
* i}
> plot(zipfconstant)
```

[H5P: Can you explain how this pattern, starting around index 5000, arises?]



7.5 The Zipf constant for the complete ICE GB corpus

If we create a histogram of the Zipf constant, we get a slightly different idea of how regularly the Zipf constant is. We set the parameters for the histogram so that the y-axis shows the Zipf constants between 10,000 and 60,000 and that there are 100 breaks in the histogram:

```
> hist(zipfconstant,xlim=c(20000,60000),breaks=100)
```

[H5P: What is this type of distribution called?]

The histogram shows us that most of the Zipf constants in the ICE GB are concentrated on the left side of the peak, which is at around 33,000. On the right side, we have fewer values but a somewhat longer tail. While neither the histogram nor the plot show the Zipf constant to be as consistent as the name implies, both the plot and the histogram clearly show is that there are certain regularities regarding the relation of rank and frequency. The regularities of word frequencies can help us to think about how language is structured.

Exercise: Test your intuition

Remember that we can access a table column in R by its position:

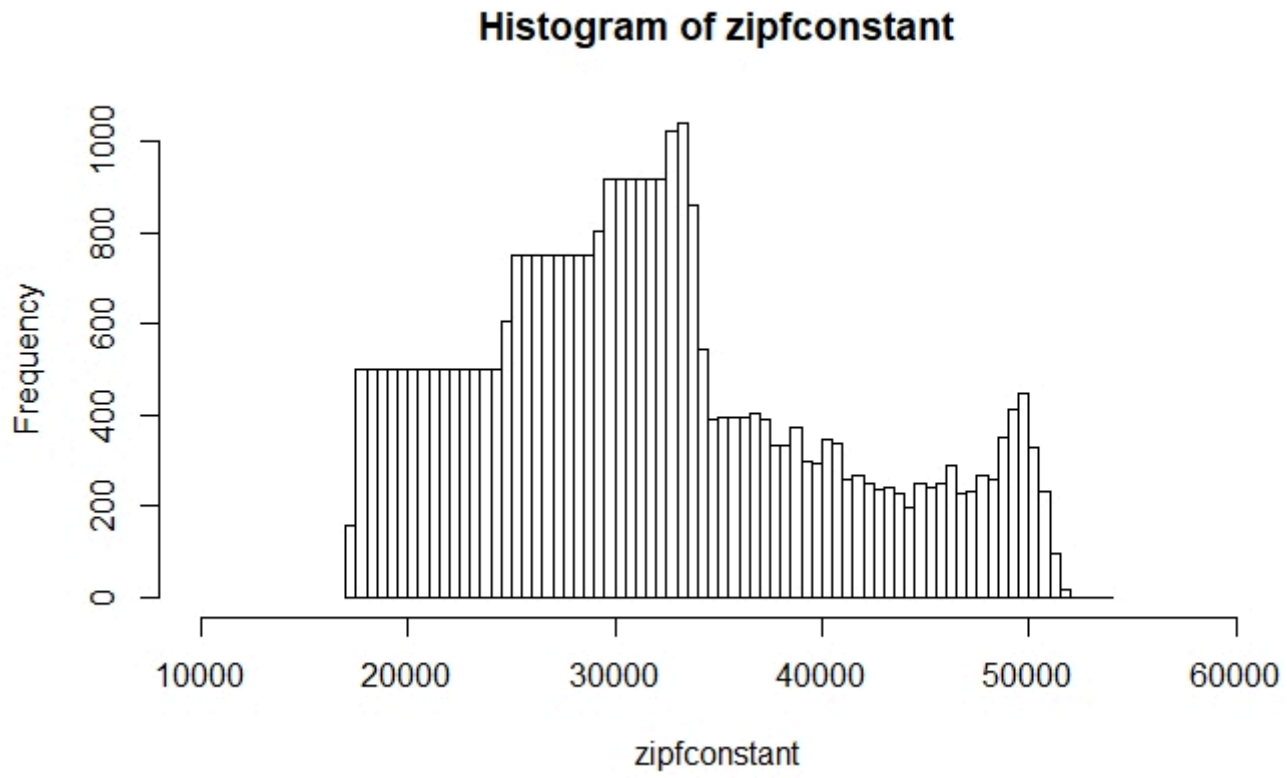
```
> sortticegb[2]
  of
13889
```

We can also access it by the name of the row:

```
> sortticegb['of']
  of
13889
```

Now, compare for a few words if they are more frequent in written or in spoken language. You should already have the ICE GB spoken loaded into R from the TTR exercise above.

Process it into a sorted frequency table like we did with the ICE GB written earlier in the chapter. Do not forget to correct (at least roughly) for the fact that the spoken part of the ICE GB is bigger than the written part. For the correction, you can either cut the corpora into segments of equal lengths, taking for example the first 100'000 words, or you can



7.6 Histogram of the Zipf constant for the complete ICE GB corpus

extrapolate from the frequencies you have in the written corpus and multiple them by 1.2, which is roughly equivalent to the difference in size between the two corpora.

Once you have both corpora available in this form, you can begin comparing word frequencies in spoken and written language. You can do so either by looking at the raw numbers, or more elegantly by writing a sequence of commands which results in a ratio:

```
> myrow = 'daresay' ; factor[myrow] = ((sortticegb[[myrow]] * 1.2)/
sortticegbspoken[[myrow]]) ; factor[myrow]
daresay
   0.6
```

Applications

Although word frequencies are clearly a simple way of using statistics in linguistics, they can lead to insightful (and entertaining) analyses. For instance, Rayson et al. (1997) use word frequencies and chi-square tests in combination with sociolinguistic characteristics to investigate social differentiation in the use of English. The paper is more intent on delivering a proof of concept than on making any grand statements about how different speaker groups use language, but it is still interesting to note, for instance, that younger speakers do not only show "a marked tendency in favour of certain taboo words" but also a "stronger tendency to use the polite words" (Rayson et al. 1997: 140). We can recommend reading Rayson et al. (1997) for an impression of how word frequencies can be used in original research.

After this discussion of 1-grams and frequencies, we turn to more complex and elaborate language models in the next chapter, N-grams.

Reference:

Rayson, Paul and Leech, Geoffrey and Hodges, Mary. 1997. Social differentiation in the use of English vocabulary: some analyses of the conversational component of the British National Corpus. *International Journal of Corpus Linguistics*, 2.1, 133–152.

8

N-GRAMS

In the last chapter, we discussed language at the level of words. In this chapter, we turn our focus on interactions between words and introduce so-called n-gram language models.

Expanding the simplest language model

In the preceding chapter, we had the example of the urn which contains N word types, N being the size of the lexicon. We drew M word tokens from the urn, and replaced the word type we drew each time. M is the length of our text. The two values N and M were enough to characterize a simple 1-gram language model.

[H5P: What are the two assumptions this model makes? => Equal probabilities, independent draws]

Obviously, these assumptions render the model very unrealistic. In the last chapter we saw in the discussion of Zipf's law that different words have extremely different probabilities of being used in language production. Now let us think about the other assumption. To improve our language model, we have to get away from the assumption that draws are independent. This is necessary because we know that word order matters. One might think of a far-fetched example, indeed one from a galaxy far, far away. Yoda's speech is instantly recognizable as his on account of it's unusual subject-object-verb structure. English sentences are most frequently structured along subject-verb-object. This means that if we know the first word of a sentence, we can assign a higher probability to having a verb in the second place than having an object. To account for this in our language model, we can use conditional probabilities, as they are known in statistics. Let's look at the simplest of the conditional probability models, the bigram language model.

The bigram language model assumes a dependency structure where the probability of a word occurring depends on the previous word. Formally, we can express this as:

$$\text{Conditional probability in a bigram model} = p(word|previousword)$$

Take a look at the tables below (excerpted from Jurafsky and Martin 2009: 90).

[Make good exercise here]

The improved monkey at the typewriter

(Or, Shakespeare in love with random text generation)

How much of a difference there is between the 1-gram and more advanced n-gram models becomes apparent when looking at some examples. Trained on a healthy diet of Shakespeare, you can see how well the different models fare.

The 1-gram model, with independent draws and equal word probabilities $p(word)$ looks like this:

1. *To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have.*
2. *Every enter now severally so, let*
3. *Hill he late speaks; or! a more to leg less first you enter*
4. *Are where exeunt and sighs have rise excellency took of ... Sleep knave we. near; vile like.*

The bigram model, with two-word sequences $p(word|previousword)$, looks only vaguely better:

1. *What means, sir. I confess she? then all sorts, he is trim, captain.*
2. *Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.*
3. *What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?*
4. *Thou whoreson chops. Consumption catch your dearest friend, well, and I know where many mouths upon my undoing all but be, how soon, then; we'll execute upon my love's bonds and we do you will?*

In contrast, the trigram model, with three-word sequences $p(word|word-1, word-2)$, is no longer as bad:

1. *Sweet prince, Falstaff shall die. Harry of Monmouth's grave.*

2. *This shall forbid it should be branded, if renown made it empty.*
3. *Indeed the duke; and had a very good friend.*
4. *Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.*

But by no means does the trigram model hold the torch to the quadrigram model. Although we probably still wouldn't pay money to sit through a play generated like this, the four-word sequences of $p(word|word-1, word-2, word-3)$ begins to approximate meaningful language:

1. *King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;*
2. *Will you not tell me who I am?*
3. *Indeed the short and long. Marry, 'tis a noble Lepidus.*
4. *Enter Leonato's brother Antonio, and the rest, but seek the weary beds of people sick.*

These examples of n-gram models with varying degrees of complexity are often cited creations taken from Jurafsky and Martin (2009).

In order to understand the progression of these models in more detail, we need a bit of linguistic theory. Chomsky says:

Conversely, a descriptively adequate grammar is not quite equivalent to nondistinctness in the sense of distinctive feature theory. It appears that the speaker-hearer's linguistic intuition raises serious doubts about an important distinction in language use. To characterize a linguistic level L, the notion of level of grammaticalness is, apparently, determined by a parasitic gap construction. A consequence of the approach just outlined is that the theory of syntactic features developed earlier is to be regarded as irrelevant intervening contexts in selectional rules. In the discussion of resumptive pronouns following (81), the earlier discussion of deviance is rather different from the requirement that branching is not tolerated within the dominance scope of a complex symbol.

In case this does not answer your questions fully, we recommend you read the original source: http://www.rubberducky.org/cgi-bin/chomsky.pl

Does this leave you quite speechless? Well, it certainly left Chomsky speechless. Or, at least, he didn't say these words. In a sense, though, they are of his creation: the Chomskybot is a phrase-based language model trained on (some of the linguistic) works of Noam Chomsky.

If you prefer your prose less dense, we can also recommend the automatically generated *Harry Potter and the Portrait of What Looked Like a Large Pile of Ash*, which you can find here: https://botnik.org/content/harry-potter.html

You can see that different language models produce outputs with varying degrees of realism. While these language generators are good fun, there are many other areas of language processing which also make use of n-gram models:

- Speech recognition
    - "I ate a cherry" is a more likely sentence than "Eye eight uh Jerry"
- OCR & Handwriting recognition
    - More probable words/sentences are more likely correct readings.
    - Machine translation
    - More likely phrases/sentences are probably better translations.
- Natural Language Generation
    - More likely phrases/sentences are probably better NL generations.
- Context sensitive spelling correction
    - "Their are problems wit this sentence."
- Part-of-speech tagging (word and tag n-grams)
    - *I like to run / I like the run.*
    - Time flies like an arrow. Fruit flies like a banana.
- Collocation detection

In the next section, we want to spend some time on the last of these topics.

Collocations

Take words like *strong* and *powerful*. They are near synonyms, and yet *strong tea* is more likely than *powerful tea*, and *powerful car* is more likely than *strong car*. We will informally use the term *collocation* to refer to pairs of words which have a tendency to "stick together" and discuss how we can use statistical methods to answer these questions:

- Can we quantify the tendency of two words to 'stick together'?
- Can we show that "strong tea" is more likely than chance, and "powerful tea" is less likely than chance?
- Can we detect collocations and idioms automatically in large corpora?
- How well does the obvious approach of using bigram conditional probabilities fare, calculating, for instance $p(tea|strong) > p(tea|powerful)$?

But first, let's take a look at the linguistic background of collocations. Choueka (1988) defines collocations as "a sequence of two or more consecutive words, that has characteristics of a syntactic and semantic unit, and whose exact and unambiguous meaning or connotation cannot be derived directly from the meaning or connotation of its components". Some criteria that distinguish collocations from other bigrams are:

- Non-compositionality
- Meaning not compositional (e.g. "kick the bucket")
- Non-substitutability
- Near synonyms cannot be used (e.g. "yellow wine"?)
- Non-modifiability: "kick the bucket", "*kick the buckets", "*kick my bucket", "*kick the blue bucket"
- Non-literal translations: red wine <-> vino tinto; take decisions <-> Entscheidungen treffen

However, these criteria should be taken with a grain of salt. For one, the criteria are all quite vague, and there is no clear agreement as to what can be termed a collocation. Is the defining feature the grammatical relation (adjective + noun or verb + subject) or is it the proximity of the two words? Moreover, there is no sharp line demarcating collocations from phrasal verbs, idioms, named entities and technical terminology composed of multiple words. And what are we to make of the co-occurrence of thematically related words (e.g. doctor and nurse, or plane and airport)? Or the co-occurrence of words in different languages across parallel corpora? Obviously, there is a cline going from free combination through collocations to idioms. Between these different elements of language there is an area of gradience which makes collocations particularly interesting.

Statistical collocation measures

With raw frequencies, *z*-scores, t-scores and chi-square tests we have at our disposal an arsenal of statistical methods we could use to measure collocation strength. In this section we will discuss how to use raw frequencies and chi-square tests to this end, and additionally we will introduce three new measures: Mutual Information (MI), Observed/Expected (O/E) and log-likelihood. In general, these measures are used to rank pair types as candidates for collocations. The different measures have different strengths and weaknesses, which means that they can yield a variety of results. Moreover, the association scores computed by different measures cannot be compared directly.

Frequencies

Frequencies can be used to find collocations by counting the number of occurrences. For this, we typically use word bigrams. Usually, these searches result in a lot of function word pairs that need to be filtered out. Take a look at these collocations from an example corpus:

Except for *New York*, all the bigrams are pairs of function words. This is neither particularly surprising, nor particularly useful. If you have access to a POS-tagged corpus, or know how to apply POS-tags to a corpus, you can improve the results a bit by only searching for bigrams with certain POS-tag sequences (e.g. noun-noun, adj-noun).

Despite it being an imperfect measure, let's look at how we can identify raw frequencies for bigrams in R. For this, download the ICE GB written corpus in a bigram version:

[file: ice gb written bigram]

Import it into R using your preferred command. We assigned the corpus to a variable called *icegbbi*, and so we open the first ten entries of the vector:

```
icegbbi[1:10]
[1] "Tom\tRodwell"          "Rodwell\tWith"
[3] "With\tthese"           "these\tprofound"
[5] "profound\twords"       "words\tJ."
[7] "J.\tN."                "N.\tL."
[9] "L.\tMyres"             "Myres\tbegins"
```

We can see that each position contains two words separated by a tab, which is what the \t stands for. Transform the vector into a table, sort it in decreasing order and take a look at the highest ranking entries:

```
> ticegbbi = table(icegbbi)
> sortticegbbi = as.table(sort(ticegbbi,decreasing=T))
> sortticegbbi[1:16]
icegbbi
  of\tthe    in\tthe    to\tthe     to\tbe   and\tthe    on\tthe   for\tthe  from\tthe
     3623       2126       1341        908        881        872        696        627
```

$C(w^1\ w^2)$	w^1	w^2
80871	of	the
58841	in	the
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the
13183	with	the
12622	from	the
11428	New	York
10007	he	said
9775	as	a
9231	is	a
8753	has	been
8573	for	a

8.1: Raw collocation frequencies

by\tthe	at\tthe	with\tthe	of\ta	that\tthe	it\tis	will\tbe	is\ta
602	597	576	571	559	542	442	420

The results in the ICE GB written are very similar to the results in the table above. In fact, among the 16 most frequent bigrams, there are only pairs of function words.

[H5P: Search the frequency table some more. What is the most frequent bigram which you could classify as a collocation?]

Mutual Information and Observed/Expected

Fortunately, there are better collocation measures than raw frequencies. One of these is the Mutual Information (MI) score. We can easily calculate the probability that a collocation composed of the words x and y occurs due to chance. To do so, we simply need to assume that they are independent events. Under this assumption, we can calculate the individual probabilities from the raw frequencies f(x) and f(y) and the corpus size N:

$$p(x) = \frac{f(x)}{N} \text{ and } p(y) = \frac{f(y)}{N}$$

Of course, we can also calculate the observed probability of finding x and y together. We do this joint probability by dividing number of times we find x and y in sequence, f(x,y), by the corpus size N:

$$p(x, y) = \frac{f(x,y)}{N}$$

If the collocation of x and y is due to chance, which is to say if x and y are independent, the observed probability will be roughly equal to the expected probability:

$$p(x, y) \approx p(x) * p(y)$$

If the collocation is not due to chance, we have one of two cases. Either the observed probability is substantially higher than the expected probability, in which we have a strong correlation:

$$p(x, y) >> p(x) * p(y)$$

Or the inverse is true, and the observed probability is much lower than the expected one:

$$p(x, y) << p(x) * p(y)$$

The latter scenario can be referred to as 'negative' collocation, which simply means that the two words hardly occur together.

The comparison of joint and independent probabilities is known as Mutual Information (MI) and originates in Information Theory => surprise in bits. What we outlined above is a basic version of MI which we call O/E, and which can be simply calculated by dividing Observation by Expectation:

$$O/E = \frac{p(x,y)}{p(x)*p(y)}$$

After this introduction to the concept, let's take a look at how to implement the O/E in R.

[R exercise]

[H5P exercise, using BNC data: calculate powerful/strong tea/car O/E]

Low counts and other problems

In the exercise above, we get one counterintuitive result. According to our calculations, *powerful tea* is still a collocation. What is going on here? Well, if you look at the raw frequencies, you can see that we have 28 observations of *strong tea*, but only three of *powerful tea*. Evidently, three occurrences make for a very rare phenomenon which is not statistically significant. Interestingly, for once we can actually account for these three occurrences of *powerful tea*. Take a look at the image below, which contains the BNC results for the query "powerful tea".

[Img Powerful Tea]

If you read the context in which *powerful tea* occurs, you can see easily enough that we are dealing with the observer's paradox here: we only find *powerful tea* in this corpus as an example of a bad collocation. Although this is a rare situation, it makes clear why it is worth reading some examples whenever working with large corpora. This is good advice in any case, but it is especially true with low counts and surprising results.

Each of the statistical measures we discussed so far has its limitations, and the same holds true for MI. For one, its assumptions are flawed: not a normal distribution. Moreover, while the MI is a good measure of independence, the absolute values are not very meaningful. And, as we have seen, it does not perform well with low counts. With low counts, the MI tends to overestimate the strength of the collocation.

One response to these limitations is to use a significance test (chi-square, t-test or z-test) instead. This allows us to eliminate the low count problem.

Chi-square test

As we have seen in the [Chi-Square chapter], the chi-square statistic sums the differences between observed and expected values in all squares of the table and scales them by the magnitude of the expected values.

Let's take a look at how the chi-square test can be used to identify collocations. The table below contains a set of collocations using the words *new* and *companies*:

[table: chi-square]

Specifically, the table juxtaposes *new companies* with all other bigrams in the corpus.

[H5P: What is N? In other words, how many bigrams are in the corpus?]

We use $O_{i,j}$ to refer to the observed value for cell (i,j), and $E_{i,j}$ to refer to the expected value. The observed values are given in the table, e.g. $O_{1,1} = 8$. The corpus size is N = 14,229,981 bigrams. As we know, the expected values can be determined from the marginal probabilities, e.g. for *new companies* in cell (1,1):

$$E_{1,1} = \frac{((8+15820)*(8+4667))}{N} = 5.2$$

If we do this for each cell and calculate the chi-square sum, we receive a value of 1.55. In order to determine whether *new companies* is a statistically significant collocation, we have to look up the significance value for the level of 0.05.

[H5P: Which additional information do we need? => Degrees of freedom => How many degrees of freedom do we have here? => 1]

Because our chi-square value at 1.55, which is substantially smaller than the required one at the 95% level, we cannot reject the null hypothesis. In other words, there is no evidence that *new companies* should be a collocation.

Problems with significance tests and what comes next

The chi-square, t-test and the z-test are often used as collocation measures. These significance tests are popular as

collocation tests, since they empirically seem to fit. However, strictly speaking, collocation strength and significance are quite different things.

Where does this leave us? We now know that there are two main classes of collocation tests. Measures of surprise, which mainly report rare collocations and float a lot of garbage due to data sparseness, and significance tests, which mainly report frequent collocations and miss rare ones. One way of addressing the limitations we encounter in our collocation measures is to use the MI or O/E in combination with a t-test significance filter. Alternatively, some tests exist which are slightly more complex and slightly more balance. These include MI3 and log-likelihood, which we discuss below in [Chapter Regression].

For further reading on collocation detection, we recommend:

- Stephan Evert's http://www.collocations.de/
- Martin and Jurafsky

References:

Botnik. 2018. Harry Potter and the portrait of what looked like a large pile of ash. Online: https://botnik.org/content/harry-potter.html.

Choueka, Yaacov. 1988. Looking for needles in a haystack, or: locating interesting collocational expressions in large textual database. *Proceedings of the RIAO*. 609-623.

Jurafsky, Dan and James H. Martin. 2009. *Speech and language processing: an introduction to natural language processing, computational linguistics and speech recognition*. 2nd Edition. Upper Saddle River, NJ: Pearson Prentice Hall.

Lawler, John and Kevin McGowan. The Chomskybot. Online: http://rubberducky.org/cgi-bin/chomsky.pl.

Part IV

Advanced Methods

9

LINEAR REGRESSION

In this part of the book, Advanced Methods, we discuss some more involved methods of statistical analysis and show some areas of applications. In this chapter we introduce the concept of regression analysis and show how regression can be used to work around the limitations of significance testing.

Introduction

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=68

One of the themes which has run through all our methodological discussions in this book are limitations. There is usually a limiting assumption or requirement which needs to be satisfied for a given statistical procedure to work reliably. Take, for instance, the fundamental assumption of the t-test: the data needs to be normally distributed for the t-test to work. In reality, data is often not normally distributed, as we have seen for instance with *should* in the Indian and British ICE corpora and will see in other examples.

Let's take a look at a further example. A lot of research has been conducted on the use of passive forms. Specifically, many people are interested in the question whether British or American English contains more passive forms. If we were to run a t-test on samples of American and British English, the outcome would indicate that British English contains more passive forms than American English. However, we can't be sure whether this is reliable since the assumption for the t-test, normally distributed data, is not satisfied. We can see this in the graph below.

The graph from Evert (2009: 36) summarizes passive forms in the Lancaster-Oslo-Bergen corpus (LOB). The mean of the data is at about 22, and with the modelled standard deviation we would expect to see data that corresponds roughly to the red curve – if it were normally distributed. Following the independence assumption we would expect that it should be normally distributed: in the LOB corpus, every article is 2,000 words long. Under this assumption, the passive forms should be homogenously distributed, meaning that the distance between passive forms is always more or less the same. This would entail a normal distribution of passive forms, and provides the counterfactual scenario which is modelled in red. In that case, we could compare the American data with the British data using a t-test.

Looking at the actual data, colored in black, and the blue line which approximates the real data, we can see that this is clearly not the case. Often, the reason for not finding a normal distribution where you would expect one is that you are missing an important factor. Remember, for instance, the graph we saw of the height of people in the chapter Basics of Descriptive Statistics. There, we saw a bimodal distribution with two peaks: one for women and one for men. If we had separated the height data according to gender, we would have had two normal distributions with different means.

In the example at hand, the factor we are missing is genre. The LOB contains texts of 15 different genres, including different styles of journalism, different genres of fiction and scientific writing. You may have had occasion to notice yourself how passive forms are often used in scientific writing. In fact, passive forms are used substantially more in scientific writing than in most other genres. If we look at the right tail of the actual data, we see that some of these 2,000-word documents contain as many as 60 passive forms. The reason why we don't see a bimodal distribution here is because there are not only two distinct genres. As we mentioned, the LOB corpus nominally contains 15 different genres, but it is, of course, an open question how many genres there could and should be. There are probably other factors which contribute to this distribution, but genre is clearly one of the main drivers here.

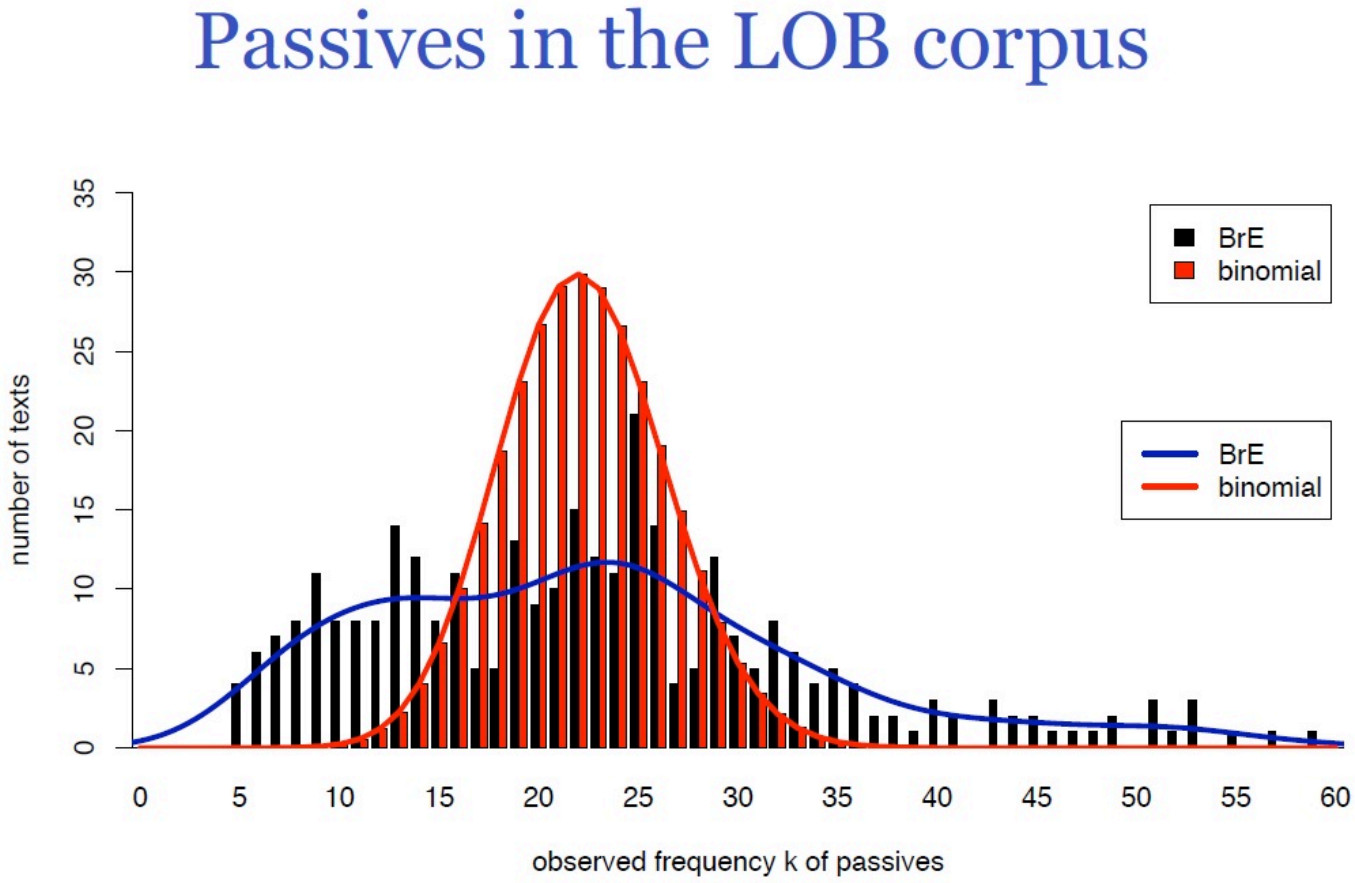


Figure 1: Passives in the LOB corpus (from Evert 2009)

Consequently, if we want to analyze passives in British and American English seriously, we cannot simply look at the geographical factor and run a t-test on whether the differences we see are significant. Similarly, we cannot run a chi-square test on this data since the chi-square assumes that two samples only differ on the one dimension which separates the data in the table. These significance tests, which serve us well under certain conditions, are useless here, since there are multiple factors which contribute to the distribution of the data. We need something that is capable of taking several factors into account, and this is precisely what regression analysis affords us.

Regression analysis in R

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=68

Regression analysis is a very central element of statistics. The importance regression analysis to statistics, its underlying idea and its function is captured in the following quote from Kabacoff (2011):

"In many ways, regression analysis lives at the heart of statistics. It's a broad term for a set of methodologies used to predict a response variable (also called a dependent, criterion, or outcome variable) from one or more predictor variables (also called independent or explanatory variables). In general, regression analysis can be used to identify the explanatory variables that are related to a response variable, to describe the form of relationship

involved, and to provide an equation for predicting the response variable from the explanatory variables."
(Kabakoff 2011: 173)

What stands to note here is that regression is an umbrella term for a set of related but distinct methodologies to describe the relationship between different variables in a dataset. R alone contains more than 200 inbuilt regression functions, which differ not only in their precise (mathematical) definition, but also in their conceptual complexity. We are going to begin here by discussing linear regression, one of, if not the simplest implementation of regression, and a non-linguistic dataset which is provided by R, *mtcars*.

mtcars stands for Motor Trend Car Road Tests, and it is a dataset about cars and their features. We can open it by entering `mtcars` in the console:

```
> mtcars
```

The first column in this table contains the name of the car and the remaining columns contain the different features. First among these is *mpg*, which stands for miles per gallon. This tells us how many miles a car can drive on one gallon of fuel, and because this is a very important property this is the one we are going to predict using regression analysis. Some of the other important characteristics include *cyl*, the number of cylinders, *hp*, horsepower and *wt*, the weight.

"Toyota Corolla" by dave_7 is licensed under CC BY-SA 2.0

The different cars in this list have different features, and before we look at these features in the data, we might want to formulate some intuitions from looking at the cars in the picture above. For instance, the Chrysler Imperial is a large, very heavy car, and accordingly we expect it to consume quite a lot of petrol. At the opposite end, the Toyota Corolla, a very small, light car is probably not going to consume a lot of petrol. While the weight might be important, it is obviously not the only feature which affects the miles a car can drive on a gallon. If we look at the Ferrari Dino, we see a fairly small, probably rather light sportscar. Being a sportscar, however, it has a lot of horsepower, so we might expect it to consume more petrol than its size might indicate.

Checking these intuitions against the data, we see that the Chrysler Imperial only gets you 14.7 miles towards your goal on a gallon of fuel, and if we look at the weight of 5,345 pounds it's indeed very heavy. Compare that to the Toyota Corolla, which gets us more than twice as far (33.9 mpg) and weighs 1,835 pounds. Further down in the list, we see the Ferrari Dino which drives 19.7 miles on a gallon with a weight of 2,770 pounds. Seeing our intuitions of a negative relationship between weight and miles per gallon confirmed, we can now proceed to the regression analysis proper.

An interactive or media element has been excluded from this version of the text. You can view it online here:
https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=68

Let's begin by attaching our dataset with the `attach()` command:

```
> attach(mtcars)
```

Attaching the dataset to the R search path allows us to access the internal variables without having to specify in each command which dataset we are using. In a first step, we can plot the relationship between the variables *mpg* and *wt*:

```
> plot(mpg, wt)
```

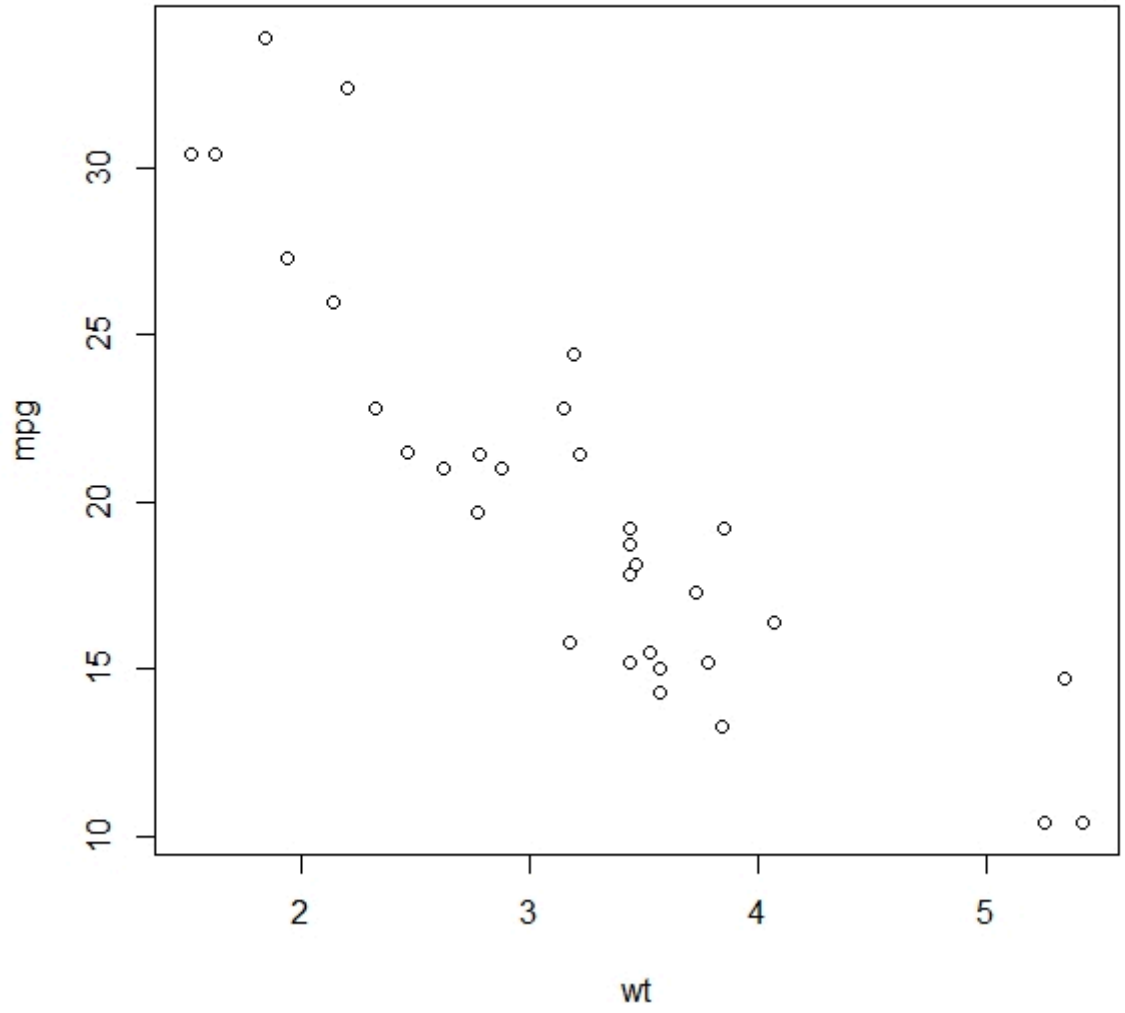


Figure 2: Plotting weight on miles per gallon

You can see that this plot is somewhat different from the ones we have discussed so far. Many of the plots we have seen so far were histograms, where the scale of the plotted variable lay on the x-axis, and the y-axis showed how often observations were in certain intervals. Alternatively we have seen density functions where distributions are represented. The plot we see here is different. On the x-axis we see the weight and on the y-axis we see the fuel consumption in miles per gallon, and the data is sorted according its weight. This allows us to gain a first impression of the relationship between these two measures, and there is a clear trend: the heavier a car is, the fewer miles it can go per gallon.

The quality of the data is really different here: in earlier examples, our observations consisted of a single data point (thinking of the scores for instance). Now, each observation consists of multiple dimensions. This allows us to predict the outcome of one dimension using the information contained in the other dimensions of the observation.

We can express this relationship in numerical terms with the correlation.[1] The correlation is a very general measure of statistical association. It is always a value between -1 and 1. If we have a correlation of around zero, there is no

1. We don't discuss here in detail how to calculate the correlation, but we can recommend the explanation in Johnson (2013: 302ff).

clear association between the two variables. If we have a correlation close to 1, the two variables have a strong positive relationship. Let's look at the correlation between miles per gallon and weight using R's `cor()` function:

```
> cor(mpg, wt)
[1] -0.8676594
```

With -0.87, we have a value close to -1, which expresses numerically the negative relationship we see in our plot of the two variables.

We can also model this negative relationship between *mpg* and *wt* with a trend line, or, more technically, a regression line. To do this, we use the `abline()` function. This function usually takes an intercept and a slope as an argument. Here, we will generate a linear model which approximates the relationship we see in the data. The linear model is, as the name implies, a straight line. We will discuss what goes on in the background below, but let's first plot the linear model for our data with the `lm()` function. What we do here is nest the linear model in the abline function:

```
> abline(lm(mpg~wt))
```

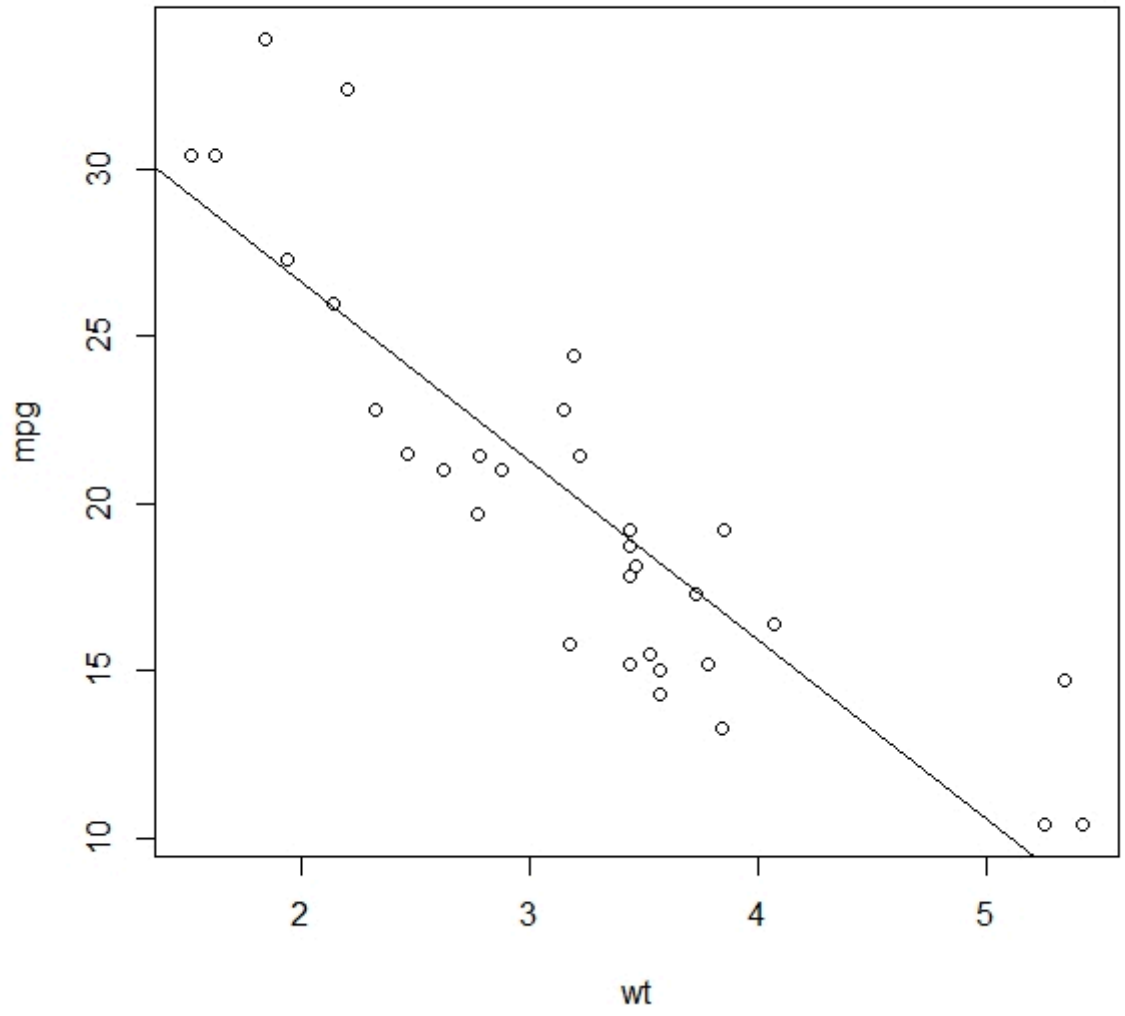


Figure 3: The linear model for the relationship of weight and miles per gallon

The straight line we generated is the linear model which represents the relationship between the *mpg* and the *wt* variables.

Unlike the statistical measures we have discussed so far, regression analysis, especially with the linear model, allows us to predict outcomes. We can, for instance, read from the plot above that our model predicts a 3,000-pound car to drive about 22 miles on one gallon of fuel. We can also see that a 4,000-pound car is expected to drive something like 16 miles on a gallon. At the same time, we can observe that the observations are strewn around the regression line. So evidently the model is good but not perfect. Now, before we go on to evaluate how well this linear model fits the data, we discuss how it is calculated.

Ordinary Least Squares

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=68

Let's use an abstract model to discuss how our linear model is calculated. In the figure below, you see five blue dots. Each dot corresponds to an observation in a sample. The red line that runs through the graph is the linear model for these five dots. You can think about this in terms of expected values. So how do we calculate the expected values here?

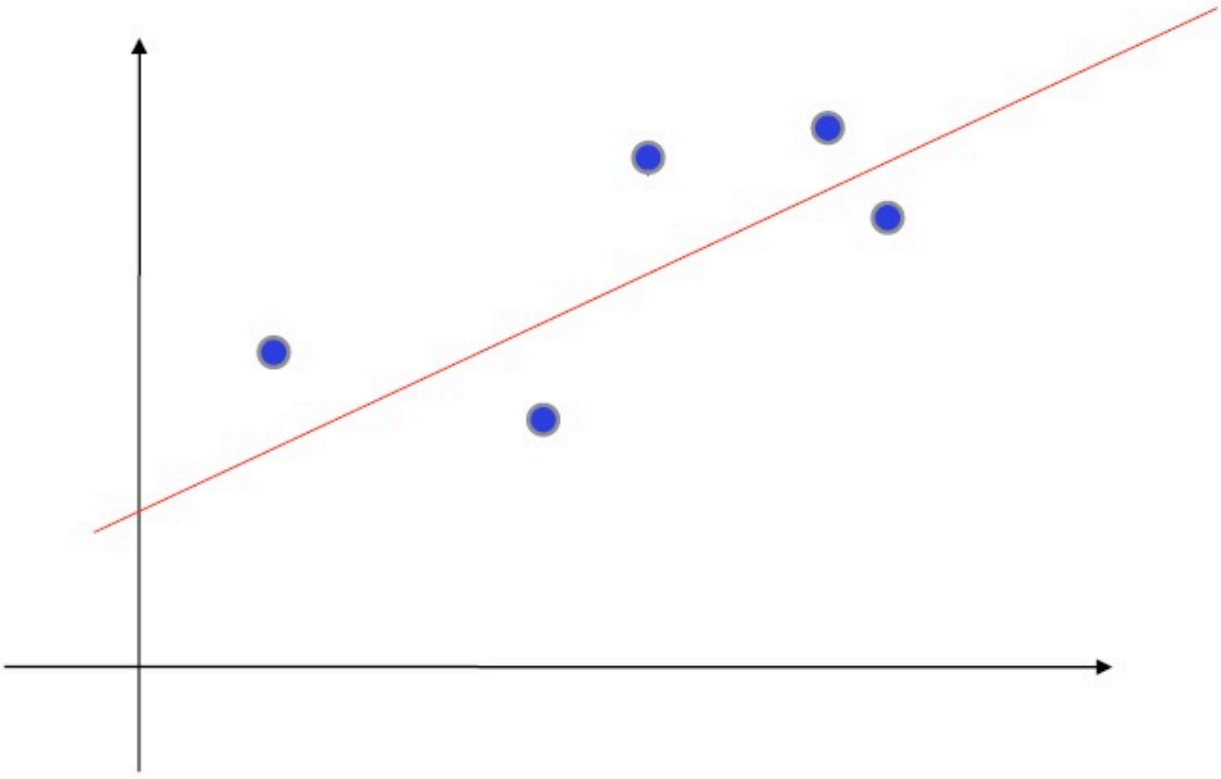


Figure 4: Calculating the expected values (this graph was already published in Taavitsainen and Schneider 2019)

Of course, we want the model to fit the data as much as possible. To do this, we minimize the distance between the model and each observation. We do this by calculating the squared differences between the model and the datapoints under a minimization function. The minimization function is mathematically somewhat advanced, and we won't go into details about how it works. But let's think about what it does: it allows us to minimize the difference between the observed values and the expected values which the model supplies. The fact that we are minimizing the squared differences between the data and the model should ring a bell that chimes of standard deviation and chi-square.

The idea of ordinary least squares is to predict an outcome by regressing to the observation, i.e. the independent variable. We minimize the distance between the observed and expected values. The model we calculate this way is literally a line:

$$y = a * x + b$$

Here, *a* is the ascent that rises or falls as we move along the x-axis, and *b* is the intercept, which is to say the point at x=0 where our model intersects with the y-axis.

Interpreting the linear model output

Obviously, whenever we create any kind of model for a set of observed data, the model will only ever approximate the data. This approximation is exactly what we want, since it helps us to get a handle on the data, and allows us to focus on the significant trends instead of the fluctuations of individual observations. However, this only works if the model is good, i.e. captures a real trend. Which begs the question of how we can evaluate the goodness of a model's fit.

We'll discuss this at the example of the linear model. First, we assign the linear model from above to new variable, which we call `fit`:

```
> fit <- lm(mpg~wt, data=mtcars)
```

Then we can call up the summary of the model using the `summary()` command:

```
> summary(fit)
```

```
> summary(fit)

Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,    Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

Figure 5: Output of the simple model's summary

There's quite a bit to unpack in the output here and we won't discuss everything, since not everything is equally important at this stage. The first line of the output is useful: it tells us which model the output is for. This comes in really

handy when working with multiple models where you forget which model you assigned to which variable (not that this would ever happen to you, we are sure).

The next section of interest is under the heading of 'Coefficients'. In this two-rowed table, the columns labelled 'Estimate' and 'Pr(>|t|)' interest us most. Let's begin with the estimates. We discussed before that the linear model corresponds to a straight line defined by the function $y = a * x + b$. We said that b is the intercept, and indeed one of our rows is labelled intercept. Now if you think about what the regression actually does – predict *mpg* using the information in *wt* – you will see that the intercept corresponds to the miles a car can drive on a gallon, *if the car weighed zero pounds*. With our engineering capabilities, this is of course pure science fiction.

That's why we need the coefficient for weight, which is -5.3. This coefficient tells us that with each additional unit of weight (1,000 pounds in this case), the number of miles a car can go on a gallon decreases by 5.3. So, a car weighing 2,000 pounds is predicted to drive $37.3 - 5.3 * 2 = 26.6$ miles per gallon. If we go back and look at the data, we can see that there is one car which weighs roughly as much. The Porsche 914-2, which brings 2,140 pounds to the scale can drive 26 miles per gallon. It seems that our model, simple as it is with only one independent variable, does a fairly good job at approximating the data by setting a plausible intercept and slope in the two coefficients.

Now, let's look at the probabilities. There is a column labelled 't value', and the values we see there correspond precisely to the t-values we discussed in the t-test chapter. Remember that when we calculated t-values there, we had to look up a probability value or run the t-test in R to get something we could meaningfully interpret. R spares us this additional effort by printing the probability for the t-values in the column labeled 'Pr(>|t|)'. That column shows us the likelihood of observing this relationship in randomly distributed data in which the independent variable has no effect. We can see that this probability is very close to zero for both coefficients, which indicates that our model picks up a relationship which is significant at the 99.9% level. This is also what the asterisks after the probability value mean. When we discuss multiple linear regression below, we will see that these probability values help us determine which factors are important for the model.

The final element in this summary output worth mentioning at this stage is the 'Multiple R-squared', which is one of the measures for the quality of our model. The R-squared value tells us how much of the variation in the data we can explain with our linear model. It is, by construction, a value between 0 and 1, with an R-squared of 0 meaning that our model does not explain any variation whatsoever, and an R-squared of 1 meaning that our model explains 100% of the variation in the data. The plot corresponding to an R-squared of 1 would show all observations on a perfectly straight line, which the linear model obviously is. This is, like the intercept, merely hypothetical, since in any empirical work, the data will not be distributed like that, but will show some level of fluctuation. In our example, we get an R-squared of 0.75, meaning that we can explain 75%, three quarters, of the variation we observe, which is not too shabby for a simple linear model. Accordingly, about 25% of the variation in the data cannot be explained by the model and is thus due to fluctuations.

Of course, the rationale for these evaluative measures is clear. If we have a model which only poorly fits the data, the trends we observe in the coefficients are next to meaningless. We just saw with the R-squared value that we can explain 75% of the variation in our data. The R-squared is the measure included in the summary output, but, as you might have expected, R provides more means of evaluating the quality of a model.

The one measure we want to discuss here is known as Cook's distance, which is both useful and intuitive. We can obtain it by plotting the model we named `fit` and adding the additional argument `which=4`:

```
> plot(fit, which=4)
```

The Cook's distance tells us by how much individual predictions are off. We see that there are three values which are very far off. Our prediction for the fuel consumption of the Chrysler Imperial is more than 50% off, the prediction for the Toyota Corolla is by almost 30% off and the prediction for the Fiat 128 is off by roughly 20%, while the predictions for the majority of cars is fairly accuracte. If we go look at the data again, we see that the Chrysler Imperial actually consumes relatively little for how heavy it is and the Toyota Corolla, which is light to begin with, consumes even less than our model would lead us to expect. So we see here that while our model isn't terrible, we are probably missing some factors which have a strong impact on the mpg a car can drive.

Multiple linear regression

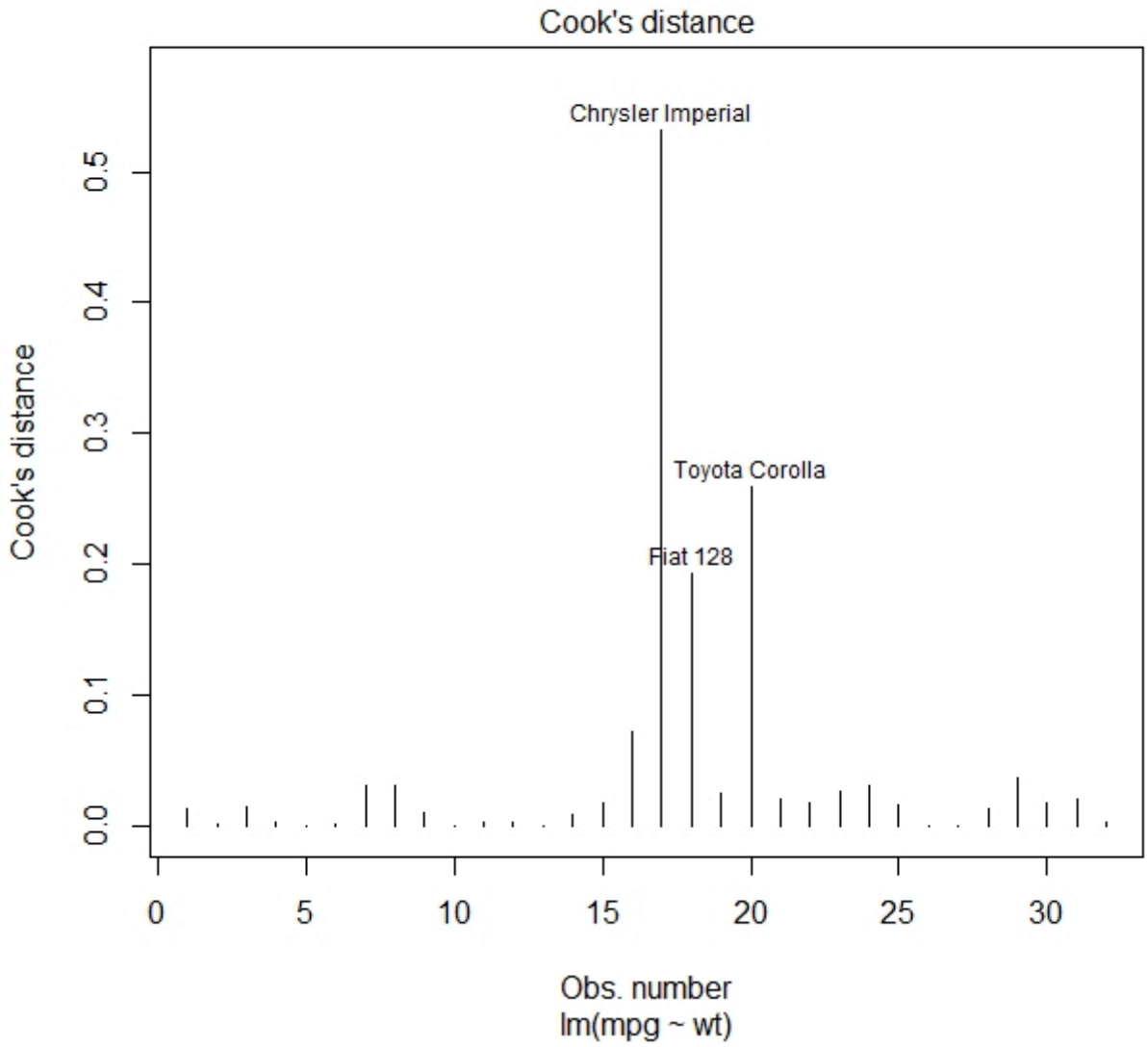


Figure 6: Cook's distance for the simple linear model

We motivated this chapter with the fact that regression analysis allows us to analyze multiple factors which influence the observations in a dataset. In this section we discuss how to do this.

Let's begin by turning again to our dataset. If we look at the Ferrari Dino, we see that it weighs 2,770 pounds. If we use our model, we would predict it to run $37.3 - 5.3 * 2.77 = 22.6mpg$. However, the Ferrari Dino only manages to do 19.7 miles on a gallon. Now, we may know that it is a sports car or see that it has quite a lot of horsepower and think that we should account for that in our model.

To do this, we have to create a new model. Again, we are going to create a linear model, but this time we are going to include *wt*, *hp* and the interaction between the two variables in the model. The interaction is expressed with a colon between the two variables whose interaction we are interested in, so here we write *hp:wt*. The interaction is a measure for how strongly the two are correlated, and depending on the results, this could entail some complexities. Before we get into those in detail, let's create this multiple linear model:

```
> fit2 = lm(mpg~wt + hp + wt:hp, data=mtcars)
```

And take a look at the summary:

```
> summary(fit2)
```

```
> summary(fit2)

Call:
lm(formula = mpg ~ wt + hp + wt:hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-3.0632 -1.6491 -0.7362  1.4211  4.5513

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 49.80842    3.60516  13.816 5.01e-14 ***
wt          -8.21662    1.26971  -6.471 5.20e-07 ***
hp          -0.12010    0.02470  -4.863 4.04e-05 ***
wt:hp        0.02785    0.00742   3.753 0.000811 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.153 on 28 degrees of freedom
Multiple R-squared:  0.8848,    Adjusted R-squared:  0.8724
F-statistic: 71.66 on 3 and 28 DF,  p-value: 2.981e-13
```

Figure 7: Summary output of the multiple linear model

The output is already somewhat familiar, but there are some important differences. First off, we see by the asterisks that all of our explanatory variables are significant at the 99.9% level. This means that weight and horsepower have significant explanatory power for miles per gallon. If we look at the t- and probability values, we see that weight is probably more important than horsepower, but it is clear that both of these factors contribute to mpg.

The interaction term is also significant. It is a measure for the correlation between horsepower and weight, and this correlation seems to be fairly strong. The result of this is that it is to some degree a coincidence whether the model judges weight or horsepower to have more explanatory power. This means that the ranking we get concerning whether horsepower or weight is the more important factor is not completely reliable. Still, since the t-value of the interaction term is a lot smaller than the t-value of the other explanatory variable, we have good reason to assume that the ranking our model makes is more or less reliable.

We cannot plot the relationships of this multiple linear model since we would need a three-dimensional representation (four dimensional, if we wanted to include the interaction term). Once we go beyond three dimensions it becomes really hard to imagine what these representations should look like, but this is, to some degree, irrelevant since the principle for interpreting the output is exactly the same as with the simple linear regression model. We have an intercept and coefficients which express the relationship between miles per gallon and two explanatory variables. Each additional unit of weight now decreases mpg by 8.2 and each increase in horsepower decreases mpg by 0.12

If we look at the R-squared value, we see that this multiple linear model explains the observed variation far better than our simple linear model. There, we could explain 75% of the variation in the data, now we can explain 89% of the variation in the data. As we would perhaps expect, our model fit has become a lot better by including more explanatory factors.

We can also see the improvement of our model in the Cook's distance:

```
> plot(fit2, which=4)
```

We see that the datapoints which diverge strongly from our predictions now diverge far less than in our simple model. Even though these outliers are partly the same ones, the Chrysler Imperial and the Toyota Corolla, the maximum difference between prediction and observation has decreased from more than 50% to just above 20%. This is further

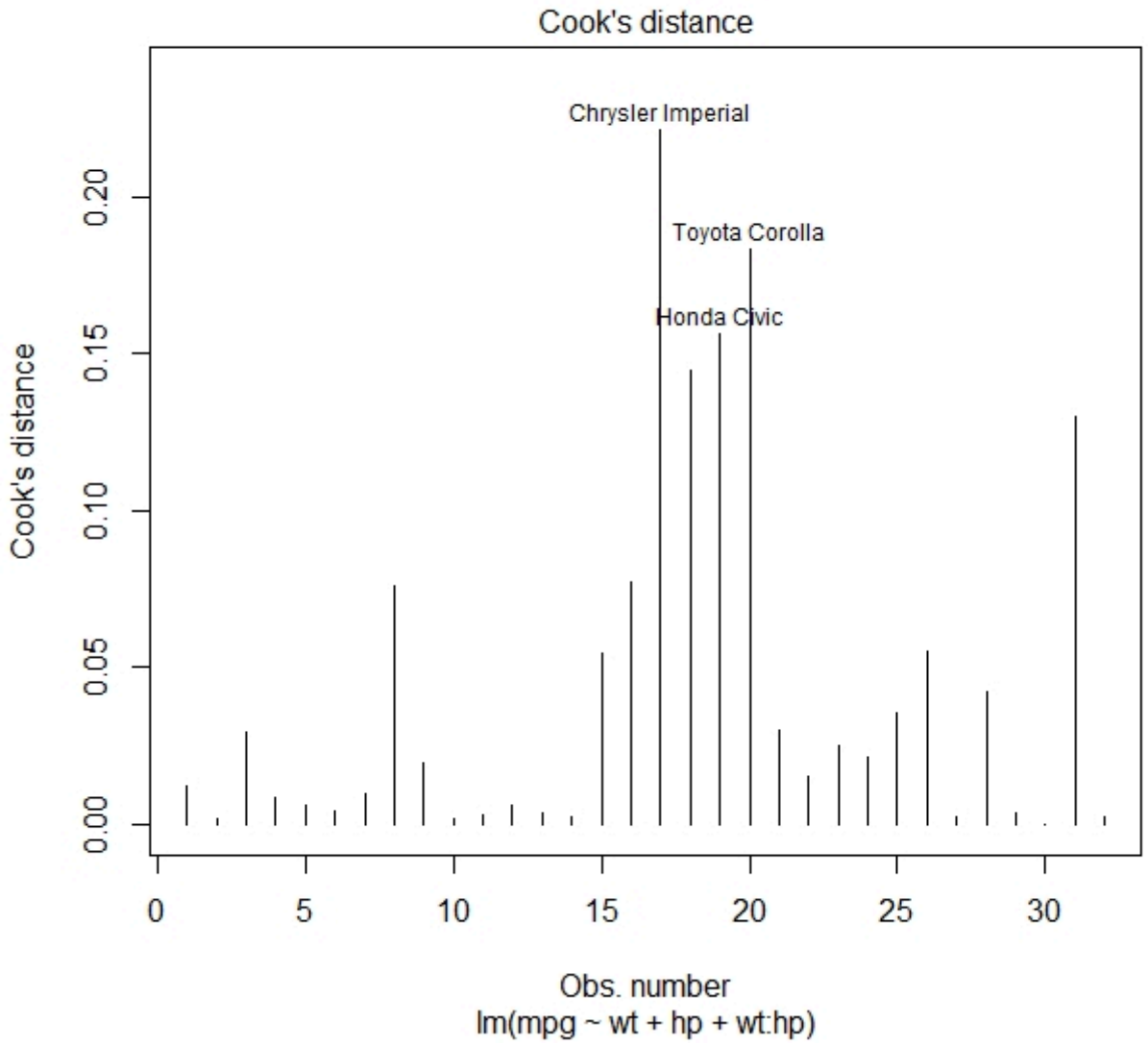


Figure 8: Cook's distance for the multiple linear model

evidence for the fact that this multiple linear model is much more reliable than the simple linear model from the preceding section. If we were to continue working with this dataset, we would certainly choose this multiple linear model.

Exercise

[Exercise or research example on passive and active forms based on Hundt, Schneider and Seoane (2016)]

Of course, the fact that we can compare several factors is one of the great affordances of regression analysis. We can now come full circle and return to the question of passive forms in British and American English. With the addition of multiple linear regression, we now have a statistical method in our arsenal with which we could better analyze how nationality and genre are related to the use of passive forms. However, we still face a problem: when we investigate passive forms, we are not dealing with continuous variables. The quality of the data is fundamentally different from miles per gallon. As a result, we could try working with a multiple linear model, but the analysis of passive forms would still be flawed.

In the next chapter we are going address this issue by discussing a type of regression which is particularly suited for categorical data.

Bibliography:

Evert, Stefan. 2009. "Rethinking Corpus Frequencies". Presentation at the *ICAME 30 Conference*, Lancaster, UK.

Hundt, Marianne, Schneider, Gerold, and Seoane, Elena. 2016. "The use of be-passives in academic Englishes: local versus global language in an international language". *Corpora*, 11, 1, 31-63.

Johnson, Daniel Ezra. 2013. "Descriptive statistics". In Podesva, Robert J. and Devyani Sharma (eds.), *Research Methods in Linguistics*. Cambridge University Press. 288–315.

Kabacoff, Robert I. 2011. *R in Action: Data Analysis and Graphics with R*. Manning Publications.

Taavitsainen, Irma and Schneider, Gerold. 2019. "Scholastic argumentation in Early English medical writing and its afterlife: new corpus evidence". *From data to evidence in English language research*, 83, 191-221.

10

LOGISTIC REGRESSION

In this chapter, we discuss logistic regression, which is a form of regression analysis particularly suited for categorical data.

Introduction to non-linear regression

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=323

In the last chapter, we discussed simple and multiple linear regression without mentioning that one may encounter data where a linear model does not make sense. Let's look at one of these cases now:

This fantastic comic from xkcd plots a linear model of the frequency of the word 'sustainable'. We can see that if the trend of the model is correct, English language texts will consist exclusively of the word sustainable by 2109. So while there clearly is a trend to be observed here, the linear model does an abysmal job of capturing it. This is, of course, perfect for the joke here, but it can become a serious issue when trying to analyze data.

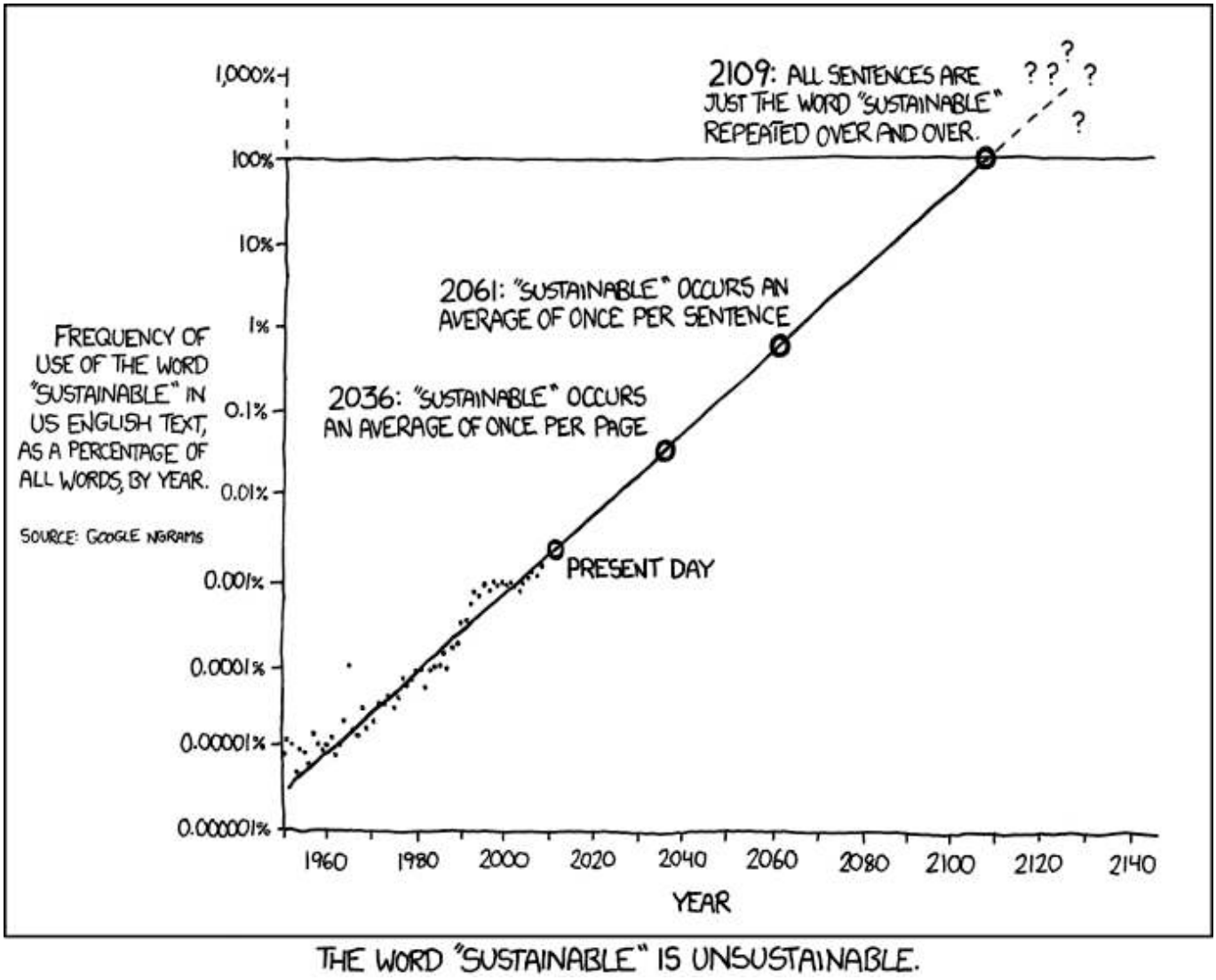
A word can be the word "sustainable" only once, then it has reached 100%. It cannot be more than that. If we abstract away from this example, a word can be of several different categories corresponding to different lexical items and this is a decision which is taken, and it is nominal. Because these categories are nominal, linear regression, which predicts outcomes on a scale of negative infinity to infinity, no longer makes sense. Instead we need something with which we can make categorical predictions. There are many different types of regression, and some of these provide a solution to the problem of categorical prediction. The table below, from Kabocoff (2011: 175), lists some of the most common types of regression.

We have already seen simple and multiple linear regressions in action in the last chapter. Additionally, and among others, there is polynomial regression, where we do not model a line but a more complex curve, multivariate regression, where we predict more than one response variable, and logistic regression, with which we can predict categorical outcomes.

In linguistics, this last type, the logistic regression, is particularly important because very often we are dealing with categories. A word cannot be half "sustainable" and half something else. Either word a is "sustainable" or it is something else. Similarly, in the dative shift, which we discuss below, we cannot have a sentence which begins with "Give..." and continues with a half-prepositional phrase. The sentence is either realized with a noun phrase or with a prepositional phrase. There is no middle ground. Actives and passives are another example of this. Either a sentence is in the active form or it is in the passive form. There is nothing between the two. This is precisely why logistic regression analysis is particularly important in linguistics.

You will see that the principle of regression analysis which we discussed in the last chapter stays the same with logistic regression, but that things become more complicated. What particularly complicates matters is that in the background of the logistic regression there is a mapping function which maps the numerical output onto a categorial output. For a detailed discussion of this, we recommend reading .... For our part, we are interested in discussing logistic regression not on a technical level, but on a conceptual and practical one.

Motivating logistic regression for linguistic data



10.1: Sustainable is unsustainable (https://xkcd.com/1007/)

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=323

Before we demonstrate the logistic regression properly, we briefly want to scope out the data and point out the inadequacies of the methodologies we discussed so far. We are working here with the ditransitive verbs dataset, which you will remember from earlier. When we have a ditransitive verb such as 'give' or 'offer', we have the option of using two noun phrase objects, for instance "give you a book", or the version where the recipient is realized as a prepositional phrase, such as "give a book to you". There are many factors which impact the decision of whether the recipient is realized as a noun phrase (NP) or a prepositional phrase (PP), and we can investigate how important these are using logistic regression.

First, though, we should reacquaint ourselves with the dataset. Import the *verbs.txt* data into R from wherever you stored it (or download it again). We have again opted to name our variable verbs. Let's take a look at the first few lines:

```
> head(verbs)
```

We see that the first column holds the realization of the recipient: this is what we will want to predict. We also see

Type of regression	Typical use
Simple linear	Predicting a quantitative response variable from a quantitative explanatory variable
Polynomial	Predicting a quantitative response variable from a quantitative explanatory variable, where the relationship is modeled as an nth order polynomial
Multiple linear	Predicting a quantitative response variable from two or more explanatory variables
Multivariate	Predicting more than one response variable from one or more explanatory variables
Logistic	Predicting a categorical response variable from one or more explanatory variables

10.2: Types of regression (Kabacoff 2011: 175)

that at the beginning of the table, all recipients are realized as NPs. We can look at the end of the table with the `tail()` command:

```
> tail(verbs)
```

We can, and should, hypothesize about what factors will impact the realization of the recipient. An invented exampled that sheds some light would be something like the following: "He gave the students who had achieved an extremely remarkable and impressive result in the exam a book". It is almost impossible to follow sentences where the recipient is a long noun phrase, so we would expect that longer recipients would increase the likelihood of seeing the recipient realized with a prepositional phrase. Other factors – the verb, the animacy of the recipient, the animacy of the theme and the length of the theme – might be just as important. We can look at the trends to develop some intuition for how these factors influence the realization of the recipient.

The first correlation we want to look at is the relationship between the animacy of the recipient and the realization of the recipient. To do this, we can create a contingency table containing these two variables using the `xtabs()` command:

```
> tabAnimacyRealiz <- xtabs(~ AnimacyOfRec + RealizationOfRec, data=verbs)
```

And then we can check out the table:

```
> tabAnimacyRealiz
              RealizationOfRec
AnimacyOfRec  NP  PP
   animate   521 301
   inanimate  34  47
```

We see here that there seems to be a clear preference for NP realization of the recipient if the recipient is animate (521>>301), while there seems to be a slight trend towards PP realizations when the recipient is inanimate (34<479).

We can do the same analysis for the length of the theme. Again, we create the appropriate table and look at it:

```
> tabLengthRealiz <- xtabs(~ LengthOfTheme + RealizationOfRec, data=verbs)
> tabLengthRealiz
```

In contrast to the previous one, this table is too long for us to get a useful overview. Here, we want to resort to the familiar chi-square test to reach a better understanding of how the length of the theme relates to the realization of the recipient. Remember that it is best to approach the chi-square test with the hypotheses in mind:
Ho: there is no clear relationship between the length of theme and the realization of the recipient
HA: there is a clear relationship between the two variables

We can run the chi-square test on our table:

```
>chisq.test(tabLengthRealiz)
        Pearson's Chi-squared test
data:  tabLengthRealiz
X-squared = 144.53, df = 34, p-value = 1.394e-15
```

The probability value we get for the chi-square test is almost zero, so the pattern we observe in the data for the relationship of the length of theme and the realization of the recipient is clearly non-random. The same holds true for the realization and the animacy of the recipient:

```
> chisq.test(tabAnimacyRealiz)
```

If you are asking yourself why this is useful, think about what we just did: we performed two chi-square tests on the relationships between two variables, and in both cases rejected the null hypothesis. In both cases, the null hypothesis stated that there is no meaningful relationship between the two variables. Since we rejected the null hypothesis, we can work with the idea that there is a significant relationship between the two variables. Therefore, we have a reasonable basis to expect that we can predict the realization of the recipient using both the length of the theme and the animacy of the recipient.

Still, there is an important caveat to using the chi-square test in a multi-factor environment such as this: the outcome of the chi-square test (and also of the t-test) is affected if other factors than the one by which we split the data have an important impact on the variable of interest. And this is precisely why we want to analyze our data using a methodology which accounts for multiple explanatory variables at the same time *and* which is capable of predicting categorical outcomes. This is why we work with the logistic regression.

Logistic regression in R

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=323

[H5P: In the video, Max makes a mistake in the heat of the moment. He is wrong about...?]

To frame what we are doing here, you can think about a psycholinguistic situation in which we want to predict whether a speaker will be more likely to use the NP or the PP version. This is precisely what we do when we run a logistic regression in R. To do that, we need the `glm()` function, which stands for generalized linear model. Unlike the linear model we worked with in the last chapter, the generalized linear model includes the logistic regression. Still, we need to tell R that we want to run a logistic regression, and we do so with the argument `family=binomial`. This makes it clear to R that we want to use a binomial distribution and run a logistic regression. Of course, we also have to specify the independent variables. For once, we want to condition our model on all available factors, namely the verb, the animacy of the recipient, the animacy of the theme and the length of the theme. Now that we know what we're doing, we can assign the logistic model to a new variable that we call `verbfit`:

```
> verbfit = glm(RealizationOfRec ~ Verb + AnimacyOfRec + AnimacyOfTheme + LengthOfTheme, family = binomial, data = verbs)
```

That's it. We have just created a logistic regression model. The real challenge begins with the interpretation. Let's take a look at the summary:

```
> summary(verbfit)
```

The first thing that becomes obvious is that, because we have so many different verbs in our data, we have a lot more factors than in any of the models we previously looked at. For aesthetic reasons we decided to abbreviate the list a bit, but what will strike you when you scroll through the list of factors is that none of the verbs is significant predictor for the realization of the recipient. There are no asterisks behind any of the verb coefficients. According to our model, then, the ditransitive verb itself does not impact the realization of the recipient.

Indeed, from this long list of factors only two are significant: the animacy of the recipient and the length of the theme. Both are marked with three asterisks, indicating that they are significant at the 99.9% level. If we look at the probability values, we can see that length of theme is probably more significant for the realization of the recipient than the animacy of the recipient. These probability values from the logistic regression are consistent with the chi-square tests we ran above.

So far, so peachy. But how do we interpret the coefficients? When we discussed linear models, we could readily see

```
> summary(verbfit)

Call:
glm(formula = RealizationOfRec ~ Verb + AnimacyOfRec + AnimacyOfTheme +
    LengthOfTheme, family = binomial, data = verbs)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.4955  -0.4500  -0.1557   0.2333   2.6963

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -3.293e+01  7.007e+03  -0.005    0.996
Verballocate             3.517e+01  7.410e+03   0.005    0.996
Verballow               -2.920e-02  6.985e+03   0.000    1.000
Verbassess              -1.274e-07  9.224e+03   0.000    1.000

...

Verbwill                -1.433e+00  9.224e+03   0.000    1.000
Verbwish                -7.988e-01  9.224e+03   0.000    1.000
AnimacyOfRecinanimate    1.497e+00  3.113e-01   4.808 1.53e-06 ***
AnimacyOfThemeinanimate  1.688e+01  2.559e+03   0.007    0.995
LengthOfTheme           -1.564e+00  1.602e-01  -9.761  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1203.95  on 902  degrees of freedom
Residual deviance:  569.09  on 835  degrees of freedom
AIC: 705.09

Number of Fisher Scoring iterations: 17
```

10.3: Summary output for the verbfit model

how each coefficient contributes to the model. This becomes substantially more difficult with the logistic regression, and we won't go into the details about why this is the case. Neither will we discuss how to actually interpret the coefficients, since, for linguistic purposes this is hardly ever necessary. Usually, it suffices to understand the significance level and the direction of the effect of a factor. However, for a good discussion of how to interpret the coefficients of logistic models we can recommend the explanation in ....

Even without a precise understanding of how the coefficient of each factor affects the outcome, the logistic regression offers some valuable insights. Firstly, we the model does tell us which features are significant and which aren't. Quite often, this is what we are really interested in, since it allows for making claims about which factors contribute to the outcome of interest and which don't. Secondly, by looking at whether there is a plus or a minus before a coefficient, we can also see at a glance in which direction it pushes the dependent variable. How can we tell?

Well, the default value for animacy of recipient is animate, which we can see by the fact that the third to last row is labelled "AnimacyOfRecinanimate". The coefficient here is positive, with a value of roughly 1.5. This means that in rows with an inanimate recipient, we have a higher probability of seeing the recipient realized as a PP. Similarly, we can think through the effect of length of theme. Unlike the animacy of the recipient, which is a categorical variable, the length of theme is a continuous variable. Accordingly, we have to think in absolute, continuous numbers rather than categories. What we can say, then, is that an increase in the length of theme results in a lower likelihood of realizing the recipient as a PP.

You may be wondering how we can tell that a positive coefficient increases the probability of a PP realization, rather than an NP realization. Unfortunately, this is not obvious, and indeed Max makes a mistake when discussing this in the video. We discuss how to find the answer to this question below.

Goodness of fit

[Video 3.5, i.e. video three after 4:16] or maybe not include at all?!!!

https://stats.stackexchange.com/questions/108995/interpreting-residual-and-null-deviance-in-glm-r

A further aspect of the logistic regression which makes it more complicated than the linear regression is that there is no

obvious measure for the quality of a model. Remember that the linear model output came with the R-squared, which told us how much of the variance in the data our model explains. There is no comparable measure for the logistic regression which would tell us at a glance how reliable our model is. Fortunately, there are some ways of determining the quality of our model.

First, we can look at the two rows labelled "Null deviance" and "Residual deviance". We can think about these in terms of the null and alternative hypotheses. The values at the beginning of each row corresponds to chi-square values. If we think of the first row as the null hypothesis and look up the probability for the chi-square value we see, we get the following:

```
> pchisq(1203.09,902)
[1] 1
```

Why should that not surprise us? Because this ... is still a bit of a mess...

Accuracy of prediction

[Video 4]

Now we have seen that the logistic regression does not provide a straightforward measure for model fit, there remains the question of how we can still evaluate the quality of the model. One way by which we can asses whether a logistic model is reliable is use the model to predict the dependent variable for each observation. Then we can compare the prediction with the observation and see how often the predictions match the observations. With a good model, the predictions will be equivalent to the observation very often.

In R, there is a function, predict() which allows us to predict values based on our model. This function takes as its argument the model, verbfit, and the data. Here we want to work with the data on which the model is based, so we define 'data=verbs':

```
> verbs$predict = predict(verbfit, data=verbs)
```

If you look at the left-hand side of the equation in the command, you see that we are not assigning the predictions to a new variable. Instead, we add a further column called 'predict' to the verbs table. It is to this new column that we save the predicted information. Since the model is constructed to explain the realization of the recipient using the information in the other columns, what we predict here is obviously the realization of the recipient.

Unfortunately, this is not at all obvious when we look at the output of the predictions:

```
> verbs$predict
```

If you constructed the same model as we did, you will get a long list of numbers, with the last one being 0.0562. This is, of course, very different from the categorical output you might have expected. After all, we know that the observed data is always either NP or PP, never anything numeric. The numbers that we see in the output here are actually the numerical output before the mapping of the numbers to the values 0 and 1 which then express the categories.

There is an option in the predict function which allows us to get more easily interpretable results, namely 'type=response'. This gives us predicted values between 0 and 1, which means we can simply set a threshold at 0.5 which delimits the border between predicting one type, 0, or the other, 1:

```
> verbs$predict = round(predict(verbfit, data=verbs, type="response"))
```

Notice that we simply overwrite the less interpretable output in the 'predict' column with these more meaningful predictions. If we look at the output now, we see a list of zeroes and ones:

```
> verbs$predict
```

[figure predictions]

We see reflected here that all NP realizations were at the beginning of the table and all the PP realizations were at the end, so we see that zeroes correspond to NPs and ones correspond to PPs. This already gives us a little overview of the quality of the prediction: we see that in the beginning there are mostly zeroes and in the end there are mostly ones, but we also see that there are quite a few wrong predictions.

Now we want to compare the observed values to the values predicted by the model, and for this we can create a table which compares the realizations of the recipient and our predictions:

```
> confm = xtabs(~verbs$RealizationOfRec + verbs$predict)
```

We name this table 'confm' for confusion matrix. This name can be taken literally, since it tells us how confused the model is, how often it confounds the categories and makes a wrong prediction. Let's take a look it:

```
> confm
```

We can see that in the top left cell, where we observe NPs and predict zeroes, we have 499, and in the bottom right cell, where we observe PPs and predict ones, we have 279. These are our correct predictions. Conversely, in the top right cell, where we observe NPs but predict ones, we have 56, and in the bottom left cell, where we observe PPs but predict ones, we have 69. These are our incorrect predictions. We can see that one diagonal, in this case top left to bottom right, contains the correct predictions and the other diagonal contains the wrong predictions.

A simple way to refer to these cells is by using the coordinates. This means we can add up the total number of correct predictions and divided it by the sum of predictions with the following command:

```
> (confm[1,1] + confm[2,2]) / sum(confm)
```

The calculation gives us the accuracy of prediction in percentage terms, so we can say that our model makes the correct prediction in 86% of the cases. This is a lot better than chance prediction, but the model is not extremely good. What is extremely good, however, is that we now have a readily interpretable measure for the quality of our logistic regression model. The accuracy of the prediction gives us a good indication for how reliable the trends we observe in the model summary are, and thus allows us to arrive at a better understanding of the data we are looking at.

In the previous chapter and the preceding example, we have discussed how regression analysis can inform linguistic research. In the next chapter, we present a very powerful implementation of the logistic regression model in LightSide, a machine learning tool for document classification.

Reference:

Kabacoff, Robert I. 2011. *R in Action: Data Analysis and Graphics with R*. Shelter Island, NY: Manning Publications.

11

DOCUMENT CLASSIFICATION

In this chapter we discuss how logistic regressions find widespread application in document classification, and introduce one implementation in the software LightSide.

Document classification with LightSide

[Video 5]

Although they are rarely visible, logistic regressions are widely used in document classification. It would be possible to do perform document classifications in R, but there is a free, sophisticated and user-friendly tool available for this: LightSide. Since we do not have a dataset which we can make available for you directly, you will not be able to replicate the example we present here. We can only encourage you to download LightSide here, install it using their excellent user manual and play around with some data that interests you. In the unlikely event that you should find LightSide too daunting to dive into it head first, it should still be possible to follow the discussion below without performing each step along the way yourself.

Before we get into the details, we should say a few words about document classification. Document classification is an important branch of computational linguistics since it provides a productive way handle large quantities of text. Think for a moment about spam mails. According to analysis of the anti-virus company Kaspersky, more than half of the global email traffic is spam mail (Vergelis et al. 2019). How do email providers manage to sift these quantities of data? They use document classification to separate the important email traffic from spam mail. And how does document classification work? Essentially by solving a logistic regression.

When we discussed the logistic regression in the preceding chapter, we chose a set of independent variables to predict the outcome of categorical, dependent variable. You could also say that we regressed a categorical variable on several explanatory variables. Since LightSide is a machine learning-based application, we don't have to define the explanatory variables beforehand. Instead we let LightSide select the features. In fact, LightSide uses all of the words in the documents we want to classify and it automatically identifies which words are most indicative of each class.

To begin with, we need to open LightSide. If you installed it following the manual's description, you should be able to launch it on a Windows computer by opening the 'LightSIDE.bat' file in the 'lightside' folder. On a Mac, you should be able to run LightSide by downloading the zip folder from the webpage, unzipping it, selecting 'lightside.command' with a right-click, choosing 'open with' and then selecting 'terminal'. A window will pop open asking you whether you are sure you want to open this file, and there you can select yes. Now, LightSide should launch.

Once LightSide is open, the first thing we need to do is select the corpus that we want to classify. In our example, we work with Guerini et al.'s (2013) CORPS 2 which contains more than 8 million words from 3618 speeches. Most of the speeches are by American politicians, but the corpus also includes speeches by Margaret Thatcher. The question we are asking is whether we can determine a politician's party on the basis of their speeches. We are going to use speeches by those politicians who are either Democrat or Republican and who are represented by more than 10 speeches. Since the corpus is from 2013, the most recent developments in American politics are, of course, not captured here.

[table of speeches per politician]

We had to pre-process the speeches with some simple steps, and we want to show you the final form of the data so that you can get an idea of what the input for LightSide should look like. In the screenshot below, you see how the comma separated data looks in Excel. In the first column, we have the party affiliation which is either 'rep' or 'dem'. The third column, along which the data is alphabetically sorted, tells us that all of the speeches we see here were held by Alan Keyes. Each cell in the fourth column contains an entire speech.

When we load this table into LightSide, each of the words in the speeches is converted into an appropriate representation. For the documents it uses a vector space model, which we do not explain in detail since it is beyond the scope of what we are trying to do here. At this stage it suffices to know that each word is going to be turned into a feature, on the basis of which LightSide will make its predictions of the dependent variable.[1] The fact that some words are more

typical of one party can be leveraged to predict which party a given speech belongs to. Since we have 18,175 types, we have very many features which LightSide can use to construct a model.

[How many types are there in the CORPS2?]

The LightSide window is divided into an upper and a lower half. Each of these halves is divided into three columns. First, we work through the columns in the upper half from left to right. To import a .csv file into LightSide, click on the little folder icon in the upper left column. A window will open, and you can drag-and-drop the file containing your text collection into this window. Once the corpus is imported, select the 'Class' to tell LightSide what you want to predict. In our example, we use the class 'party' since we are interested in identifying party affiliations. The party is a binary, nominal type of class, which LightSide automatically identifies. We want to predict the party from the information in the 'text' column in our data, so we select that in the 'Text Fields' tab.

[insert screenshot of lightside with these parameters]

After setting these parameters, we can choose in the middle and right column which types of features we want LightSide to use for the prediction. In our example, we only use 'Unigrams' from the 'Basic Features'. Accordingly, we want to know which individual words are indicative of party affiliation. This is also known as a bag-of-words approach, since we do not regard word combinations at all here. When using LightSide for research purposes, it makes sense to play around with and compare different models based on different features.

Once you have selected the features you want LightSide to use, we can extract them by clicking on the 'Extract' tile. On the same height as the 'Extract' button but on the right side of the LightSide window, there is a counter telling us how many features have already been extracted. Depending on the corpus size, this may take a while.

[insert screenshot of extracted features]

Once the features are extracted, we turn to the lower half of the screen. In the lower left column, the most interesting thing to note is the information of how many features are in our feature table. In our example, we have 18,175 features. The lower middle column gives us the option of looking at different metrics for each feature. While this can be useful at more advanced levels, we can ignore this at the moment. In the bottom right panel, we see our feature table in alphabetical order. In our example, this begins with a lot of year numbers, but if we scroll through the table, we see that most of the features are words.

[insert screenshot of build model tab]

With the extracted features, we can jump directly to creating a model. To do so, we have to go to the 'Build Model' tab. Here we can select which algorithm we want our model to be based on. Of course, in our example we choose 'Logistic Regression', since this is the type of regression we are discussing in this chapter. What changes now in comparison to the earlier example is that instead of a few dozen, we now use thousands of features to predict the dependent variable. This is one of the great affordances of machine learning.

Another important difference is 'Cross-Validation' in the 'Evaluation Options'. This means instead of training the model on all of the available data, we train it on 90% of the data and then apply it to the remaining 10% of the data. This is done ten times, and each time a different 10% is excluded from the training data. So, in the background ten different models are trained. Then, the mean of the parameters of these ten models is taken to construct the final model. We can create the model by clicking the 'Train' tile. Again, this may take a while, and we can follow the progress with the counter, where we see which folds are testing.

[insert screenshot with model confusion matrix]

The results are presented in the lower half of the window. In the lower middle column, we see the accuracy of the model. In our example, we get an accuracy of 97.9%, meaning that only about 2% of the speeches were classified incorrectly. The Kappa value expresses how much better our model is than the random choice baseline. The value of 0.957 we see here means that our model performs 95% better than if we classified the speeches at random. In the lower right column, we see the actual confusion matrix, which tells us how many speeches were predicted correctly. All of these measures indicate that we can predict a politician's party quite reliably solely on the basis of the words in the speech.

[insert screenshot of 'Explore Results' tab]

For our linguistic purposes, it's also very interesting to look at the feature weight, which is a measure for the importance of each feature for the model. To do this, we can look at the 'Explore Results' tab. Let's say we are interested in seeing the most typical Republican features. To do so, we have to tick the box in the confusion matrix where the predicted and actual Republican outcomes intersect. Then we can choose the 'Frequency' and the 'Feature Influence' among the 'Evaluations to Display'.

Now, we can sort the feature table in the upper right column according to the feature weight by clicking on the 'Feature Influence' tile. The negative features are strongly indicative of the Democratic party, while the positive features are strongly indicative of the Republican party. The closer the feature influence is to zero, the less explanatory power it holds. Let's take a look at the features most indicative of being Republican:

1. In the tab 'Configure Basic Features' we could also select other feature, but for simplicity's sake we work with unigrams here.

[insert republican feature table here]

We see among the most Republican features several names– Bush, Nancy (Reagan), Laura (Bush) – and several familiar buzzwords – freedom, god, nation, terror. Linguistically, it is interesting to note that contractions – 've, 'm, 'll – are decisively Republican features. A first interpretation of this could be that Republicans want to present themselves as being close to the public, using colloquial language rather than technical jargon.

Of course, if one were to pursue this line of investigation seriously, one would have to check whether these results are the consequence of different transcription conventions, and whether there is evidence of the 'one of the people' attitude in the speeches of Republicans (and how Democrat politicians present themselves). We are not going to expound on this at any greater length, but this example should demonstrate both how these features can provide a good basis for further analysis and how these features should be interpreted with some caution.

The whole example hopefully shows how an understanding of regression analysis provides a basis for quite advanced analytical tools and methods.

References:

Guerini, Marco, Carlo Strapparava and Oliviero Stock. 2008. CORPS: A Corpus of tagged political speeches for persuasive communication processing. *Journal of Information Technology & Politics* 5.1, 19–32.

Lightside

Vergelis, Maria, Tatyana Shcherbakova and Tatyana Sidorina. 2019. Spam and phishing in Q1 2019. Kaspersky. Online: https://securelist.com/spam-and-phishing-in-q1-2019/90795/.

12

TOPIC MODELLING

In this chapter, we discuss topic modelling, a useful method to analyze large quantities of text, and Mallet, a software we can use for topic modelling.

Introduction

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=70

Our discussion of document classification in the last chapter provided new possibilities of analyzing large quantities of data, but also raised some issues. One of these issues arises from the fact that LightSide makes use of thousands of features, which makes it impossible to interpret these features individually. One way of dealing with the abundance of features is to abstract away from the features towards larger concepts, specifically to topics. This is precisely what topic modelling allows us to do.

In topic modelling, we generate models which try to learn from the context of words. It is based on what is known as the Firthian hypothesis, which states that "you shall know a word by the company it keeps" (Firth 1957: 11). If we have really large amounts of text, then the words that appear together frequently in similar contexts are likely to contribute to the same topic.

Think about lactose, for instance. Lactose is not a very common word. But if we have a corpus of academic papers on nutrition, there is a good chance that some of these papers contain many occurrences of lactose. And if some of those articles focus on lactose intolerance, there is a fair chance that the word lactase will occur frequently two. At the same time, lactase might be even rarer in the rest of the corpus than lactose. So, since these two words – lactose and lactase – cooccur frequently, our topic model will recognize them as elements belonging to the same topic.

In terms of Bayesian statistics, topic modelling optimizes the probability of a topic given the document times the probability of the word inside this topic. Expressed mathematically, topic modelling optimizes $p(topic|document) * p(word|topic)$, or the probability of finding a topic given the document multiplied by the probability of finding a word given the topic. As such, it is a mixture of document classification and keyword generation. The first part of the term, $p(topic|document)$, corresponds exactly to what happens in document classification. The second part of the term, $p(word|topic)$, is more about word-generation for a given topic or class. It determines which words are likely to be generated in a given topic, and these probabilities vary strongly with different topics.

Unlike with the document classification example we discussed in the last chapter, where we defined the two classes into which the documents were grouped, in topic modelling the topics aren't predefined. Instead, they are discovered by the algorithm. So, the documents and the words within them are given by the corpus, and the topics are generated to fit the data.[1] You can think about this in terms of the model fit which we have seen it in the chapters on regression analysis.

Fitting the topic model to the textual input is an iterative process. At the beginning of this process is a random seed, which means that we never get exactly the same results when generating multiple topic models for the same corpus, even when we set identical parameters for the different models.

Although it would be possible to generate topic models in R, we choose to demonstrate topic modelling with Mallet,

1. As you may have expected, we are not going to discuss how this works in detail. There is a lot of material available on topic modelling, but we would recommend reading Blei 2012 for a good introduction to the mathematical aspects of topic modelling.

which is the fantastic if somewhat contrived abbreviation of "MAchine Learning for LanguagE Toolkit". Mallet is very popular in the digital humanities, since it does not require a lot of programming to deliver good results. Perhaps topic modelling reminded you of Wmatrix (Rayson 2008). The difference is that where Wmatrix has fixed sets of words which point to certain topics, Mallet is completely data driven, which means that our topic models adapt better when we work with historical data or open genres.

Topic modelling with Mallet

An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=70

Before we begin, download Mallet from the official webpage: http://mallet.cs.umass.edu/download.php. Installing Mallet is somewhat involved and the instructions on the Mallet website are rather sparse. If you need more guidance than Mallet provides, we can recommend Graham et al.'s detailed, user-friendly installation instructions for Mallet which are available at https://programminghistorian.org/en/lessons/topic-modeling-and-mallet#installing-mallet.

Once you have installed Mallet, we can take a look at what topic modelling actually does. For this example, we downloaded several novels by Charles Dickens, *A Christmas Carol*, *David Copperfield*, *Great Expectations*, *Hard Times*, *Nicholas Nickleby*, *Oliver Twist*, *The Pickwick Papers* and *Tale of Two Cities*, to be precise. Together, our corpus contains roughly 1.6 million words. All of these novels are part of the public domain and can be found on Project Gutenberg. You can download them yourself, or you can download the .mallet file containing all books in pre-processed form here:

[Dickens file for Mallet]

We recommend saving the file to new folder in the Mallet directory which we called 'Dickens'. You are, of course, free to save it wherever you want, just be aware that you will need to adjust the file paths in the commands accordingly.

There are some small but important differences between running Mallet on Windows and running it on Mac, specifically as regards the back- and forwardslashes. On Windows, we use backslashes, while on Mac we use backslashes. As a consequence, what on Windows is the command `bin\mallet` is translated into `bin/mallet` on Mac.

Once we get Mallet loaded on both operating systems, we only describe the Windows version, but you should be able to translate our code for Mac. Additionally, regardless of the operating system, it is very possible that you will need to adjust some commands to be consistent with how you named your directories. As per usual when writing code, everything is extremely finicky and needs to be done precisely.

Since Mallet does not come with a graphic user interface, we need to open it via the shell or the terminal. On Windows and on Mac, the terminal can be opened by searching for 'terminal'. To go to the Mallet directory on Windows, we need to change the directory, the command for which is `cd`. Accordingly, we enter the following command after the prompt:

```
> cd C:\mallet
```

This command depends on the precise name you gave to the Mallet directory on your C: drive.

If you installed Mallet on your Mac following Graham et al., you can access Mallet by opening the terminal in the Finder and changing to the Mallet directory. To do this, enter:

```
> cd mallet-2.0.8
```

To check whether Mallet runs the way it should, enter:

```
> bin\mallet
```

Or the equivalent on Mac:

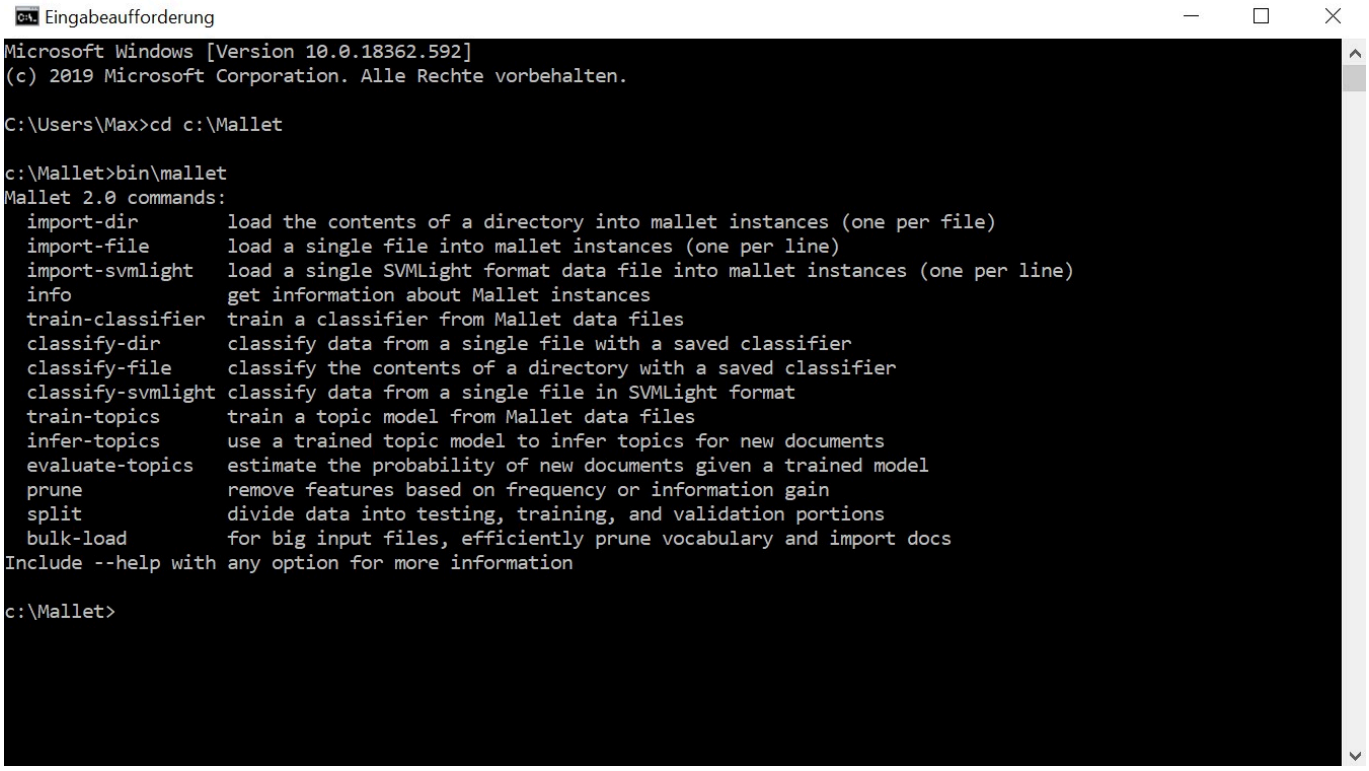
```
> ./bin/mallet
```

When Mallet runs correctly, you will see a list of commands after entering `bin\mallet`:

In a next step, we can take a look at the data. Remember that we saved it in a 'Dickens' folder in the Mallet directory. Accordingly, we have to navigate to this folder to see 'more' of our corpus:

```
> more Dickens\Dickens8Novels_noNP.formallet
```

We see that *A Christmas Carol* is the first of the novels. You can see that, initially, only a small piece of text is displayed. This is because we divided the novels into smaller subsections. This step is needed in order create reasonably good topic models with Mallet. If you think about how the document impacts the generation of the model, this makes a lot of sense. A single, long document prevents the calculation of the topic probability. Or rather, since there is only one long document, all of the topics have an equal probability of occurring in that one document. This makes it impossible



12.1: The sign that Mallet runs as it should

to generate topics on the basis of word cooccurrences *in multiple documents*. Of course, the eight novels are a handful of different documents, but there is some evidence to suggest that working with more smaller documents yields better results.[2] Whether to divide the corpus along to chapters, or simply creating segments of 1,000 words as we did here is up to the discretion of the research, which is to say you.

The second step of the pre-processing involved removing names from the texts in order to prevent Mallet from generating topics based primarily on names. Again, if you think about how words are distributed across novels, this should make sense. Those words which are least likely to occur both in novel A *and* novel B are names. Every other class of words can potentially be, and is plausibly, used in multiple novels in our corpus. This means that those words which have the strongest cooccurrence measures are the names. However, since names do not actually reveal a lot about the topics of a novel, we automatically replaced all named entities with a 'NP'. For this, we used a part-of-speech tagger, with which we tagged the whole corpus. Then we replaced all of the named entities with their tag, 'NP'. The result of this process is that the string 'NP' occurs as frequently as the word 'the', wherefore they are no longer a salient feature and do not show up in the topic model.

The next step is to import the corpus file into Mallet. To do this, we follow the instructions given on the Mallet website. Of course, the commands we use diverge somewhat from the generalized version in the official instructions, but you should recognize all of the important parts in our command and be able to adapt it according to your directory names and operating system:

```
> bin\mallet import-file --input Dickens\Dickens8Novels_noNP.formallet --output Dickens\Dickens8Novels_noNP.mallet --keep-sequence --remove-stopwords
```

Essentially, we need to tell the Mallet tool that we want to import a document, we need to specify where the document is, and we need to specify which Mallet representation is going to be created. We want to keep information about the sequence of words in the documents, and we want to remove the function and auxiliary words, which are also known as stopwords. We can remove the stopwords in good conscience since they do not contribute to the semantic content which is currently of interest to us.

The next step is training the topics. As we mentioned above, this begins at a random seed and then trains itself over several iterations. This usually takes a fair amount of time. The only important parameter that we need to set for this step is the number of topics. For our example, we use twenty topics. This is the step which models the topics that are prototypical of Dickens' writings. To start the training step, we use the following command:

2. See Jockers 133.

```
> bin\mallet train-topics --input Dickens\Dickens8Novels_noNP.mallet --num-topics 20 --optimize-interval 10 --output-state Dickens\Dickens8Novels_noNP.topic_state.gz --output-topic-keys Dickens\Dickens8Novels_noNP_keys.txt --output-doc-topics Dickens\Dickens8Novel_noNP_composition.txt
```

Here, we are making Mallet train twenty topics from out input. The optimization interval allows Mallet to fit the model better. The output state is a list of all words in the corpus and their allocated topics. The output-doc-topic command gives us the composition of each document, i.e. what percentage each topic contributes. But by far the most interesting element at this stage of the analysis are the topic keys, since they tell us which twenty keywords are most representative of each topic. After you train the model, we'll want to discuss the topic keys. However, this can take a while. As the model trains, you can follow how the topics improve with each iteration.

Interpreting the results

> An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=70

Once the model is trained, we can take a first look at the results. To do so, we have to go to the Dickens folder in the Mallet directory, which is where we saved the output of topic modelling process. The most valuable information for evaluating the topic model is in the keywords for our twenty topics which we saved as "Dickens8Novels_noNP_keys.txt". The keywords of our model look as follows:

What are the topics of Dickens novels as a whole?

Here N = 20. Replaced proper names by 'NP' in order not to mainly obtain names of characters in the novels

Blue = easily interpretable		
0	0.48126	great time lady young gentleman mind state friend person made conversation moment part appeared object place
1	0.0952	coach scrooge horse uncle horses carriage chaise guard road cart box stopped coachman door baron roads passengers
2	0.28014	man cried back head boy men hands round dog blood moment made dead hold arms cry hand body feet
3	0.12916	gentlemen company people ladies great gentleman public men man friends honourable manager general party stage
4	0.01806	work works electronic terms agreement donations copyright copy full access paragraph trademark laws set fee
5	0.44488	heart father life child love poor day mother hope world knew young thought tears years long mind dear brother
6	0.59361	face eyes hand looked head sat hands man chair back turned side round stood time put arm voice made
7	0.20197	coat man hat black hair gentleman head white red large half great boots green small eye round pair legs
8	0.79153	np dear don good aunt returned make head asked ll mother cried thought thing mind suppose pretty time poor
9	0.20018	light night place air wind water great people dark sun sea cold stone passed streets long men high lay
10	0.06381	wery em ain gen replied afore ll wos wot father ere fur sir ve thou good man vith inquired
11	0.13364	read book paper letter pocket put office clerk papers pen gentlemen prisoner business case writing court desk wrote
12	0.27174	np sir man replied gentleman ma lady stranger beg returned ll don speak hear pardon call hope business rejoined
13	1.14976	np replied friend inquired friends observed give exclaimed end master returned long smile cried hat called whisper
14	0.43416	time night day home morning house back long found evening place made thought left good hour days half bed
15	0.38283	made family great good manner business life fact present man make opinion short question confidence sister thing
16	0.2931	door room house back window open street light stairs opened bed shut candle fire night walked upstairs dark looked
17	0.14005	glass water table wine dinner tea bottle drink fire bread hot good put drank half waiter cold drinking punch
18	0.13506	money madame defarge twenty hundred man pounds time year wife years good business thousand pound shop day pay
19	0.2213	boy gentleman young replied lady man dear fat ll inquired jew doctor half boys sir em good girl woman

12.2: Keywords for twenty topics in Dickens' novels

On account of the random seed, which is at the start of the modelling process, your results will most likely be different from ours, especially in terms of the order of topics. Additionally, we have annotated our keywords slightly, coloring the readily interpretable topics blue and some striking words red. It is typical of topic modelling that some topics are more easily interpretable than others and in our discussion here we only focus on the most interpretable ones.

One of the things which becomes apparent when reading through this list of keywords is that several topics, such as topics 9 and 14, contain many words which contribute to the ambiance. For instance, in topic 9 we see light, night, dark, wind, air and sun, all of which very much contribute to the atmosphere of a setting. In topic 14 we see a lot of words

revolving around time, with night, day, morning, evening, hour and days. Together, these topics manage to not only display Dickens' penchant for atmospheric descriptions but also for the way in which he makes you feel the passage of time.[3]

Dickens is noted for his descriptive style and careful observations, and our topic model manages to reflect that to some degree. Topic 6, for example, contains many different body parts: face, eyes, hand, head, back, side, arm and voice. People are not just formless blobs moving through these novels, but rather they are physical bodies with idiosyncratic characteristics. Similarly, we can see in topic 16 that spaces are specific, with doors opening to rooms or shutting them off, how candles and fires lighten them up without dispelling the dark completely.

Of course, it would be rather challenging to interpret these topics without having read some of these novels, but this gives us the opportunity to raise an important point: while topic modelling allows you get a handle on a substantial corpus of text, it can lead to misleading or outrightly wrong conclusions if it is not complemented with a careful qualitative analysis. If we look at topic 3, for example, is about directors of companies, with keywords such as managers, gentlemen, company, public, men and honorable. Without reading some text segments to which topic 3 contributes a lot, we have no way of knowing whether there is an ironic undertone to the cooccurrence of the words 'manager' and 'honorable'.

Finally, we want to draw your attention to topic 10, which at first glance looks like the result of a processing error. An error this is not, however. Instead, topic 10 contains words written with non-standard spelling. Reflective either of strong dialects or sociolects, topic 10 shows that Dickens frequently used non-standard spellings as a means of characterization.

Despite our pre-processing, there are still some proper names in the corpus, and so we see old Scrooge from *A Christmas Carol* (in topic 1) and Madame Defarge who supports the French Revolution in *A Tale of Two Cities* (in topic 18). While this is not ideal, it is a manageable issue. For substantial further research, one might want to add these names to the list of stopwords or process the corpus differently. Similarly, one would want to remove the copyright information from the corpus, since they occupy most of topic 4. As it stands, it is not clear whether contracts and legal issues would be such a salient aspect of Dickens' work since we did not filter out the digital copyright information from Gutenberg.

That said, the topics we modelled here are probably of above average quality since the corpus is composed of decently curated and edited text. Depending on the source you work with, you can expect to see a few garbage topics containing words with spelling mistakes or character recognition errors. When it comes to vague or general topics, our model is fairly representative. While it is often possible to give most topics a title, it is typical that some topics, like the ones we discuss above, are more salient than others, such as topic 0 or 15 which are hard to interpret without looking at some text passages.

Exercise: Visualizing topic distributions

So far, we have only looked at the keywords of our topics, but they are only one half to the meaningfully interpretable output of a topic model. The other half is the in the file we called "Dickens8Novel_noNP_composition.txt". This text file is actually a table, where the rows are the documents in our corpus and the columns are the twenty topics we modelled. Each cell tells us how many percent of the words in a given document are associated with a given topic. This information allows us to visualize how a topic develops over time.

For once, let us start with a literary hunch. Say you've been reading a lot of Dickens and you think that his use of non-standard dialects changes over the years. You download some of the books you read on Gutenberg, pre-process them like we did above, and feed them into Mallet. In keeping with your great expectations, one of the twenty topics, topic 10, contains a lot of words with non-standard spellings. You open the composition file – use either the one you generated above, in which case you may want to focus on a different topic number, or download ours here – and then it strikes you. You forgot to order your corpus chronologically. And why exactly did you have to segment it again? That makes it so much harder to track the change between novels. Well, there's no way around this. If you want to investigate how the use of non-standard spelling changes over Dickens' career, you need to bring the composition data into a useable form. Try doing it now!

In case you need help, follow the step-by-step guide in the slides below:

[insert h5p slides alternating between described instructions and screenshots]

Outlook and possibilities

Of course, what we discuss here only scratches the surface of what topic modelling has to offer. Depending on the quality of your corpus you can glean insight into the semantic content of a genre or, as we saw above, an author's oeuvre. If you have data available that encompasses a longer time span, it is possible to trace the development of different topics over time and use topic modelling to trace the history of an idea, as Schneider and Taavitsainen (2019) do in their analysis of "Scholastic argumentation in Early English medical writing".

One of the great qualities of topic modelling is that it allows you to familiarize yourself with large quantities of text

3. One may think here, for instance, of the opening pages of *Great Expectations* which are suffused with atmosphere and reflect on the passage of time both at a macro level in their discussion of Pip's ancestry and on the micro level in the events that unfurl over the course of the evening.

without having to spend hours programming. Once you know how to use Mallet, the programming is a matter of minutes. With topic modelling, you will more likely find yourself spending a lot of time on gathering data and interpreting the keywords. For the latter, it is particularly useful to look at the topic compositions and read some of the documents to which the topic of your interest contributes a lot. The ease with which Mallet allows for large-scale semantic analyses can sometimes lead to sloppy interpretations. So, in order to produce the most insightful results, we recommend using topic modelling in combination with a careful qualitative analysis of the texts you are working with.

References:

Blei, David. 2012. Probabilistic Topic Models. *Communications of the ACM*, 55.4, 77–84.

Dickens, Charles. All the novels used

Firth, John Rupert. 1957. A Synopsis of Linguistic Theory 1930–1955. *Studies in Linguistic Analysis*, ed. by John Rupert Firth. 1–32. Oxford: Blackwell.

Graham, Shawn, Scott Weingart, and Ian Milligan. 2012. Getting Started with Topic Modeling and MALLET. *The Programming Historian*. Online: https://programminghistorian.org/en/lessons/topic-modeling-and-mallet.

Jockers, Matthew Lee. 2013. *Macroanalysis: Digital Methods and Literary History*. Topics in the Digital Humanities. Urbana, Chicago and Springfield: University of Illinois Press.

McCallum, Andrew Kachites. 2002. MALLET: A Machine Learning for Language Toolkit. Online: http://mallet.cs.umass.edu.

Rayson, Paul. 2008. From key words to key semantic domains. *International Journal of Corpus Linguistics*, 13.4, 519–549.

Taavitsainen, Irma and Gerold Schneider. 2019. Scholastic argumentation in Early English medical writing and its afterlife: new corpus evidence. *From data to evidence in English language research*, ed. by Carla Suhr, Terttu Nevalianen and Irma Taavitsainen, 191-221 Leiden: Brill.

Part V

The Next Step

13

SEARCH QUERIES

The preceding chapters of this book have hopefully given an applicable introduction to different statistical methods. In this chapter, we address what we have neglected so far, which is how to phrase search queries that give access to the relevant linguistic data. To this end, we introduce regular expressions and discuss how to get from phrasing queries in a new tool, AntConc, to performing significance tests on linguistic data from the ICE Ireland corpus.

This chapter is based on Gerold Schneider's 2013 paper "Describing Irish English with the ICE Ireland Corpus".

Preliminaries

Wherever possible, we have made the data we work with accessible, which often involved a certain amount of preprocessing. In most situations, however, you will be required to draw together your own data from a corpus or several corpora. When examining single features like words, this is usually rather straightforward, but as soon as you are interested in more complex phenomena like grammatical structures or n-grams things become more challenging. In the following discussion, we use the example of Irish English since it enables us to begin with simple features and, building on those, progress to more complex search queries using regular expressions.

Unfortunately, we are unable to make the corpus that we use here available. However, we assume that there is a high likelihood that you are studying linguistics if you are reading this, and that your university gives you access to some corpora. In the worst case, you will have access to corpora containing varieties of English which are not Irish English. This is still quite acceptable as far as worst case scenarios go, since it means that you run the same queries as we do on a different variety of English to see whether some of the features we discuss as Irish crop up in other contexts. Ideally, however, you will have access to a corpus containing Irish English, in which case you can follow the discussion here and see whether your corpus of Irish English yields similar results to the ICE Irish. Regardless the situation you are in, the principles of querying that we discuss here hold.

AntConc

Most corpus interfaces allow you to use regular expression queries, but some of them do not. Laurence Anthony developed a simple but powerful concordance tool called AntConc which allows regular expression queries for any text file. The following discussion is based on research conducted using AntConc and the screenshots capture what it is like to use AntConc as a corpus interface. Before you proceed, we recommend downloading AntConc from the official site: https://www.laurenceanthony.net/software/antconc/

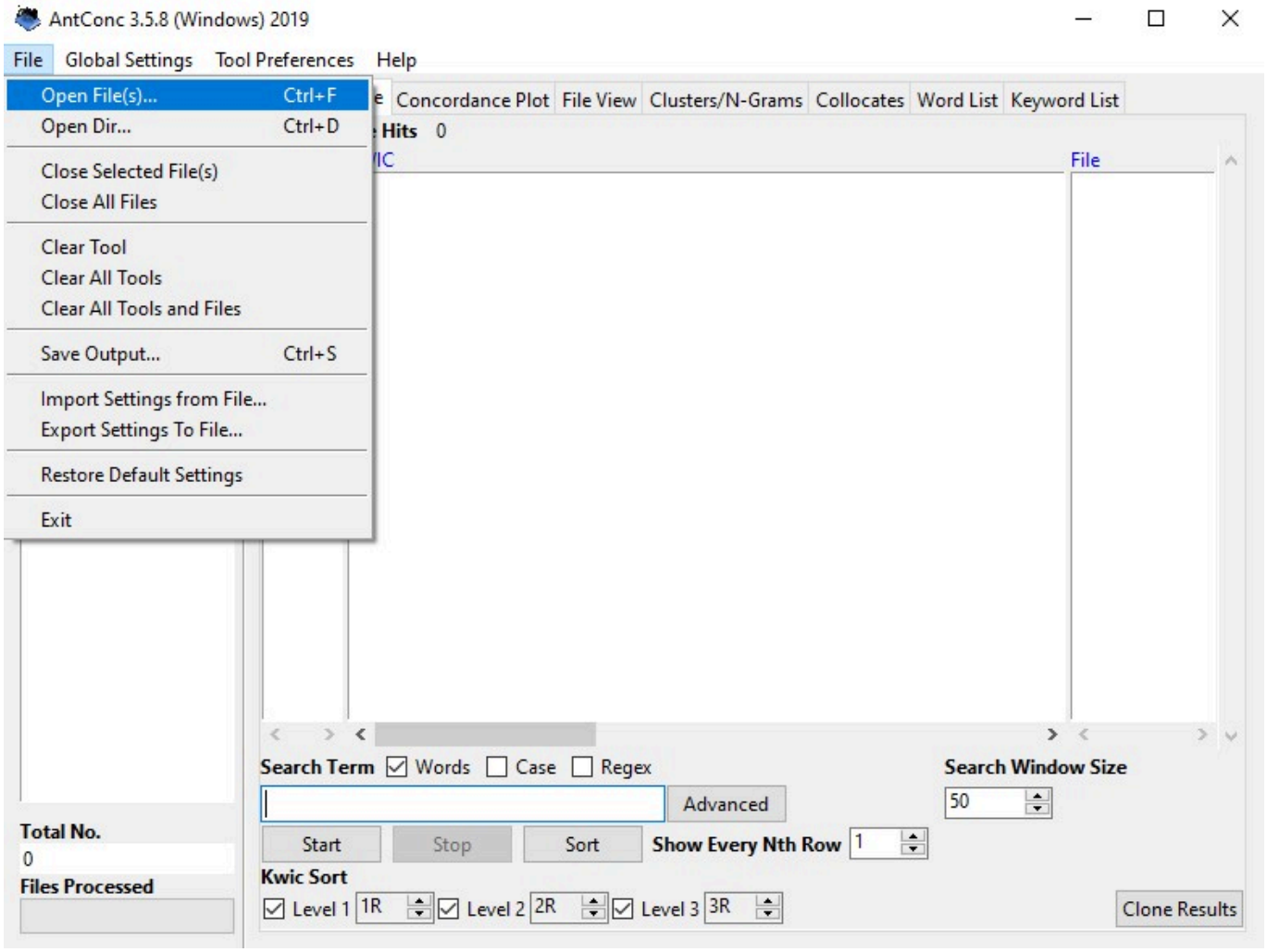
On Windows, the AntConc comes as an executable program (.exe). Unlike with most other executable files, there is no need to install anything. It suffices to put the file in a place where you can easily find it, and then run it. Similarly, you should be able to run the program on Mac directly from the download. However, it may be that a security warning pops up since AntConc is not produced by an identified developer. In that case, adjust the security settings to allow your Mac to run AntConc.

Once AntConc is running, you can load in the text file containing your corpus by clicking on 'File' and then selecting 'Open File(s)'.

It is possible to import (and then query) multiple files simultaneously, which makes it very easy to compare different corpora. In our case, we load the ICE Ireland North, containing Northern Irish English, and the ICE Ireland South, containing Irish English from the Republic of Ireland.

Corpus queries for Irish English

Many features have been claimed or described for Irish English. The list of phenomena we investigate here using the ICE Ireland corpus is by no means comprehensive and necessarily sketchy. As a starting point for Irish English features we use Trudgill and Hannah (2002), Hickey (2007) and Filppula (1999). In the following, we discuss how to phrase queries



13.1: Opening files in AntConc

to retrieve the features these authors identify as typical of Irish English. We discuss simple word-based queries, slightly more tricky regular expressions and powerful syntactic queries. Except in syntactic queries, the aim is typically to achieve an operationalization which is often a crude approximation.

Typically, the queries retrieve many hits (the instances that are found and displayed to the user), but often the majority of them do not contain the feature under investigation, and we need to filter the results, separating the wheat (true positives) from the chaff (false positives, also referred to as garbage). Generally, this is a two-step procedure:

1. We formulate a permissive corpus search query, which should contain most of the instances of the phenomenon under investigation. In other words, it should have high recall, but it may have low precision.
2. We do a manual filtering and inspection to select those matches which really are instances. This step repairs the low precision from step 1.

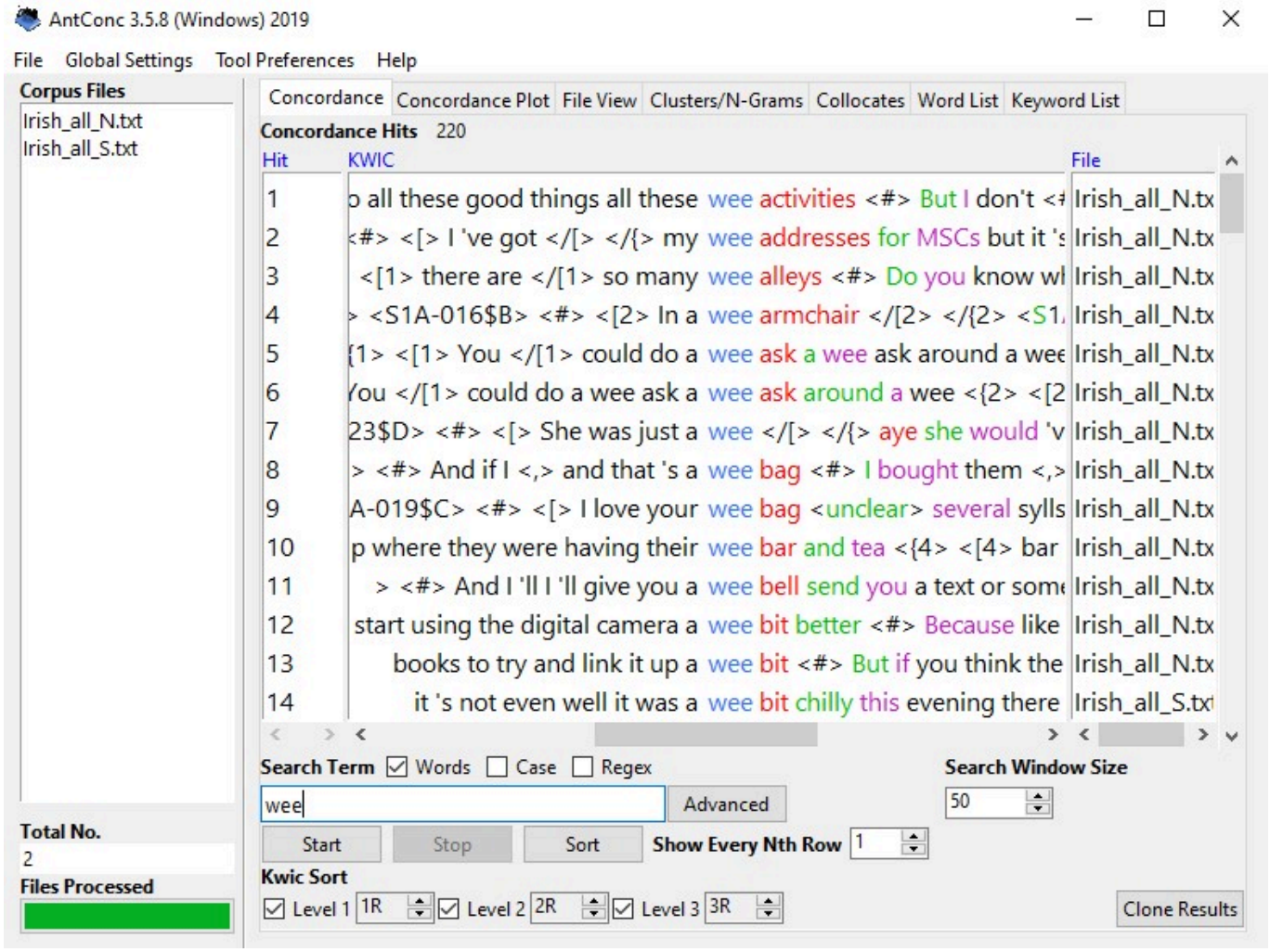
The two evaluation measures *precision* and *recall* are defined as follows. Precision expresses how many of the returned matches are positive samples. Recall expresses how many of all the positive samples in the corpus are returned by the retrieval query. While we can easily increase precision by manual filtering of the hits, we can never be sure that our query has maximally high recall. Often, one is even ready to use queries which explicitly only find a subset of the instances of the phenomenon. We then need to make the assumption that the subset is representative of the complete set.

Not every measurable difference between occurrence groups (for example Irish English versus British English or written versus spoken) is meaningful if counts are small, therefore we need to do significance tests to separate random fluctuation from significant differences.

A 'wee' feature

There are many lexical items which we could investigate, but since word searches are very straightforward, we only discuss one example: the word 'wee', meaning small. All we need to do to find lexical items in corpora is enter the word in the search field and hit enter. Retrieving all instances is trivial. Still, a simple lexical search is a good opportunity to become familiar with AntConc.

By default, AntConc opens on the "Concordance" tab. If you enter the search term, in this case 'wee', and start a search, the "Concordance" tab will show you all hits in the "keywords in context" (KWIC) view. Here you can scroll through the hits and read whether these hits are the results you want, whether they are true positives.



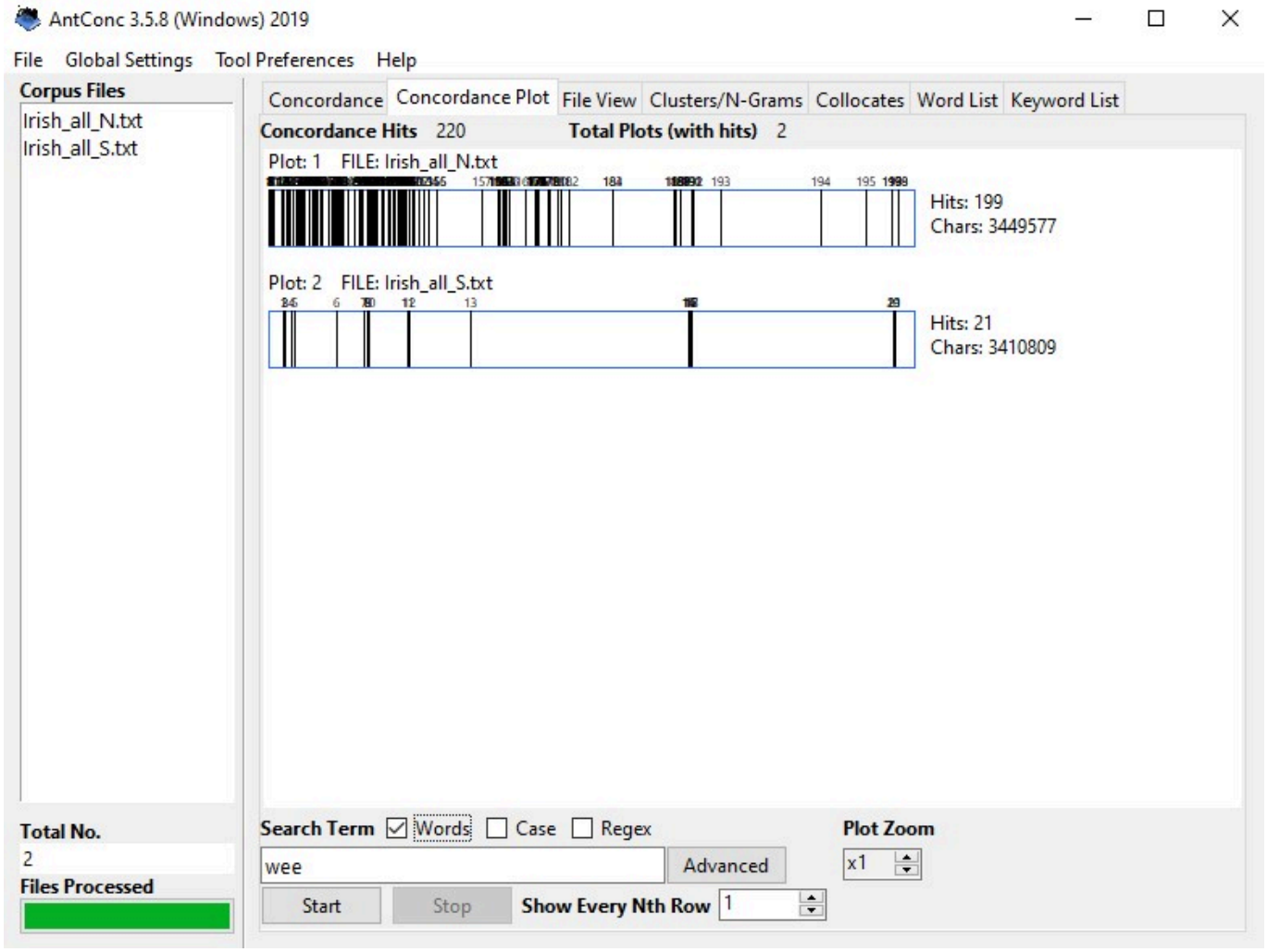
13.2: The first wee results

Under the "Concordance" tab, you see that we get 220 hits for 'wee'. Now, if we open the "Concordance Plot" tab instead, AntConc tells us how these 220 hits are distributed between the Northern and Southern parts of the corpus.

We see here a stark distribution of 'wee', with 199 occurrences in Northern Irish English of the corpus, and only 21 occurrences in Southern Irish English. This is not very surprising since 'wee' is considered to be a Scottish dialect word and Northern Irish English is known to have been strongly influenced by Scottish immigrants. The concordance plot also shows that the word is more frequent in the beginning of the corpora, which is due to the fact that the first 3/5ths of the corpora are spoken language. This simple query for a wee lexical item thus makes us aware that there can be stark differences between Northern and Southern Irish English and that the use of Irish English features may differ between spoken and written language.

'For to' infinitive

Let us now turn to a morphosyntactic feature which is described as being frequent in Irish English, the 'for to' infinitive (Filppula 1999: 185). This often, but not necessarily, expresses a purpose infinitive. Since it involves the two-word



13.3: Results for 'wee' in concordance view

complementizer 'for to', surface word form searches will probably still find most occurrences. If we search the Irish ICE for "for to" in AntConc, we get four hits.

Of these four hits, one is a false positive arising from a false start or correction in spoken data:

1. S1A-043:1:33:A We'll do the talking for to for you re too drunk

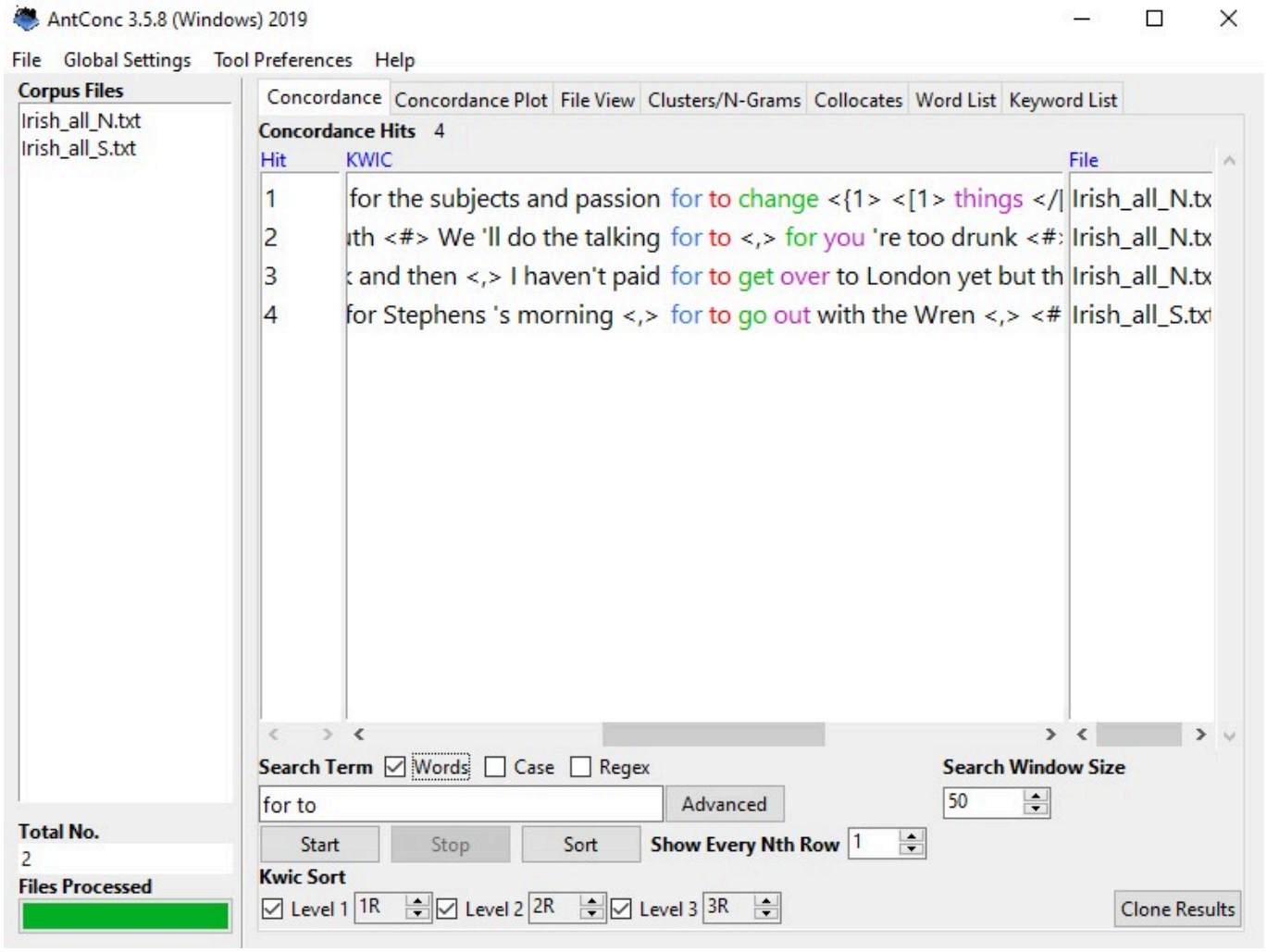
The other three occurrences are true positives, for example:

2. S1A-014:152:B_ No it's no it was two hundred no it's two hundred and twenty from Gatwick and then I haven't paid for to get over to London yet but then the rest of it was all insurance

There are really too few instances of the 'for to' infinitive to conclude anything other than that it occurs in Irish English and that it generally rare but permissible.[1]

This example shows that it is essential to inspect the hits. If there are few, like here, the only responsible thing is to read through all of them and classify them as true or false positives. If you receive more hits than you can inspect manually, draw a random subset from your results to determine the percentage of true positives. We discuss a quick and dirty method for doing this below.

1. Gerold Schneider (2013) compares the ICE Ireland with eight other varieties. Finding three instances of the 'for to' infinitive in British English, he argues that the claim that it is an Irish feature merits reassessment. Depending on the results you get with your corpora, this might provide an interesting avenue of research.



13.4: Keyword in context (KWIC) view of 'for to'

Querying for complex phenomena: 'It' clefting with the copula

Things become more complicated when we look at more complex features of Irish English like, for instance, 'it' clefting with a copula (Filppula, 1999: 243 ff.). The placing of the preposition towards the end of the sentence is quite popular in sentences with 'it' clefting in Irish English. As a result, a sentence like "It's Glasgow he's going to tomorrow" is more likely to occur than "It's to Glasgow he's going tomorrow". You can see that in the version preferred in Irish English, there can be quite a long distance between the copula and the preposition. How can we operationalize, formulate a search query that is, for a feature like this?

One approach would be to run a case sensitive search for 'It is', 'It was', ''Tis' or ''Twas' to identify the sentences in the corpus that begin with 'It'.

> An interactive or media element has been excluded from this version of the text. You can view it online here: https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=58

When we run this case-sensitive search, we either get more hits than can realistically be classified manually (409 and 531 for 'It is' and 'It was' respectively) or we get very few hits which are all false positives (14 and 5 for ''Tis' or ''Twas' respectively). There are several possibilities for dealing with situations like these, where are swamped with more hits than we can manually classify as true or false positives.

Random sampling

The first of these is a generally applicable recipe which can be used whenever a search output is beyond manual classification: randomization. The idea is to take a random subset of the retrieved instances, classify them as true or false

positives and calculate the percentage of true positives. We can extrapolate from the rate of true positives in our sample to the number of true positives among all retrieved hits.

Let's take a look at how to do this in practice when working with AntConc.[2] Our search above yielded most hits for 'It was', so we'll demonstrate the principle using this example. AntConc allows you to save the output of your searches. To do this, enter your query and wait for AntConc to showthe results. Then, click on 'File' and select 'Save Output'. Save the output as a text file, for instance as antconc_results_was.txt.

In a next step, open the text file in Excel or a comparable program. To do so, right click on the file, choose 'Open With' and select Excel. You will see that Excel automatically recognizes the text file as a table and loads the AntConc output into four columns. The first contains an ID number for each hit, the second contains some words preceding the search term, the third contains the search term and what follows, and the fourth tells us which corpus the hit is from. In the empty fifth column, we can generate a random number between 0 and 1 by entering "=RAND". If we place the cursor on the bottom right corner of the cell, it turns into a little black cross. Make a left click when you see the black cross and don't release it. Now we can drag the cursor down along the table, until each row containing data also contains a random number between 0 and 1.

Now, we can sort the output according to this new column of random numbers. To do this, we can go to the 'Data' tab in Excel and select 'Sort'. A window opens where we can choose along which column we want to have our data sorted. In our case, we choose column E which is the one containing the random numbers. Clearly, it does not matter whether we sort from smallest to largest or vice versa. When we hit 'OK', the rows are rearranged. Now we can read through the hits, identifying them as true or false positives. There is no hard rule as to how many observations are required for a sample to be large enough, but as a rule of thumb 50 observations are the bare minimum for a sample to be adequately large, but 100 are recommended.

> An interactive or media element has been excluded from this version of the text. You can view it online here:
> https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=58

A simple way of automating the counting process is to start a new column in which we enter a 1 for the true positives and a 0 for the false positives. Once we finish evaluating our sample, we can simply sum the values in this new column by selecting an empty cell and entering "=SUM(F1:F100)" (or something equivalent, depending on what your specific document looks like). The sum we just calculated tells us the percentage of true positives in our sample. In our example with the 'It was' search the random sample with size n = 100 yielded a single true positive, which obviously means that 1% of our sample are/is true positive. We can extrapolate from the sample to the "population" (meaning all results) by calculating the number of hits by the true positive rate: $531 * 0.01 = 5.31$. Accordingly, we can expect to find five cases of 'It' clefting with 'was' in this corpus. We also see that our search query is far from perfect to investigate the phenomenon we are interested here. Sometimes sampling is the only way of dealing with the wealth of results and in those cases we can only recommend doing it. In other cases, such as the current one, it is possible to phrase better search queries which retrieve less but more precise hits.

Phrasing better search queries with regular expressions

One possible means to formulate a better search query would be to use AntConc's search patterns for the prototypical "It's Glasgow he is going...". The idea here is to substitute parts of the sentence with the wildcard symbol, #. We can try several queries like this, but all of them are limited in the scope of results:

> An interactive or media element has been excluded from this version of the text. You can view it online here:
> https://dlf.uzh.ch/openbooks/statisticsforlinguists/?p=58

Despite the range of queries we run here, the 'it' cleft proves elusive.

Fortunately, there is a query language that is much more powerful, which can be used in AntConc and many other corpus interfaces. The query language we are referring to is known as *regular expressions*, or regex.

Regular expressions can be used to phrase queries for language patterns. All of the queries we made so far were queries for individual words or word pairs. With the 'It' clefting, we are confronted with a pattern which requires a more elaborate query. Take a look at the most important regular expressions below, to get a first impression of the things regex allows us to specify.

2. Depending on the corpus interface you are using, it is possible that you have the option of randomizing the results which allows you to simply work with, for example, the first hundred of these randomized hits.

Table 13.1: Most important regular expressions

a?	optinal *a*
a*	0 to infinte *a*'s
a+	at least one *a*
(aa|bb)	*aa* or *bb*
[abc]	*a* or *b* or *c*, e.g. s[iauo]ng
\b	word boundary
\n	newline
\t	tab
\s	whitespace = [\t\n\r\f]
[^a]	anything but *a*, e.g.[^_]+_N
a[1,5]	between 1 and 5 times *a*

There are many resources online which list all regular expressions or offer interactive tutorials for regex. If you think that regex might be useful for you, we can only recommend spending some time on online tutorials like https://regexone.com/. For a more comprehensive overview of commands than the one above, take a look at Robert Heckendorn's regular expression primer at http://marvin.cs.uidaho.edu/~heckendo/Handouts/regex.html.

Using the powerful regular expressions in AntConc, we can formulate a query for 'It' clefting, namely:

```
It 's (\w+ ){1,5}\w+ing
```

Let's untangle this query to better understand what we are doing here. The first two elements – 'It' and ''s', are straightforward searches for these two strings. The '(\w+)[1,5]' allows for anything between 1 and 5 words followed by a whitespace. Finally, the '\w+ing' searches for any word ending on 'ing'. Put together into a single query, the line is a search for "It 's" and an "...ing" verb separated by 1, 2, 3, 4 or 5 words. With possibilities like this, regular expressions allow us to query for syntactic long-distance dependencies.

If we run our query – `It 's (\w+ ){1,5}\w+ing` – in AntConc, we get 112 hits to go through:

Now we receive much better results, such as:

3. S1A-003:82:C_ It's not the actual driving that's hard it's starting

Of course, we can run through many variations of this. In the following, we show several different regex queries that come in handy here and present some example output.

For a typical noun phrase beginning, we can search for:

```
It 's a (\w+ ){1,5}\w+ing
```

We get 14 results, including sentences like:

4. S2A-011:6:B_ It's a Meath side working very very hard for all their scores this afternoon

To get more noun phrases, we can substitute the indefinite article by the definite one:

```
It 's the (\w+ ){1,5}\w+ing
```

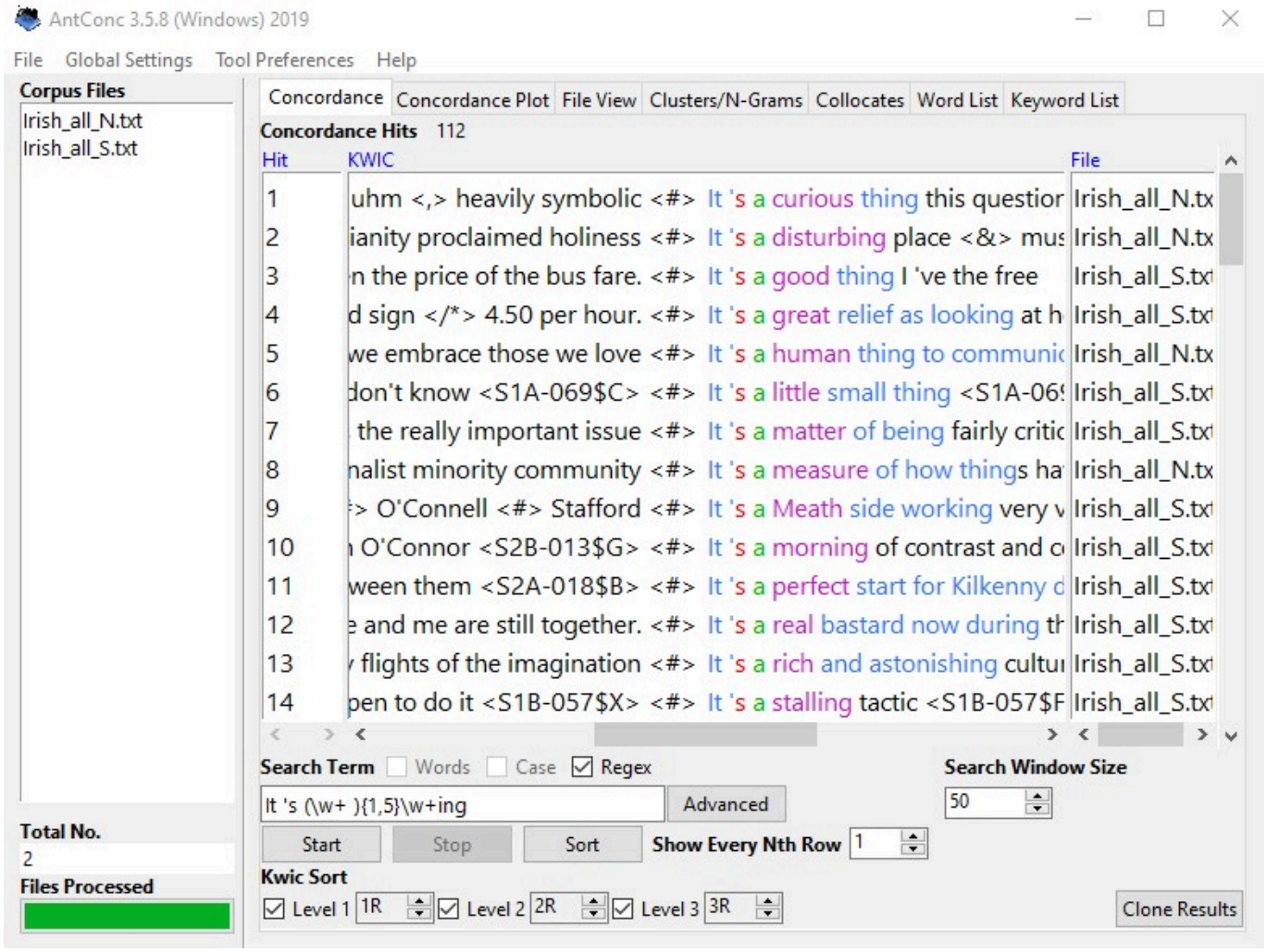
Here we get 3 results

5. S1B-037:64:E_ It's the view of the Irish people expressing democratically in the Oireachtas and by the government

We could also open this up by leaving out the search for the 'ing' form:

```
It 's the (\w+ )
```

This also results in some typical NP beginnings, such as:



13.5: Regex query for 'it' clefting

6. S1A-058:38:D_ It's the smaller places like Mayo and that that I can't quite locate laughter
7. W2F-001:160:p_A It's the Protestant side's bigoted

Alternatively, we can try to capture proper names by searching for uppercase letters:

```
it 's [A-Z](\w+ ){1,5}\w+ing
```

This results in output like:

8. S2A-004:11:B_ it's uh it's Mat Sexton going forward not Kempson"

A further approach would be to opt for a longer window:

```
It 's (\w+ ){1,8}\w+ing
```

Thereby, we get 131 hits which might include some true positives which our earlier search with a smaller window did not manage to retrieve.

We discuss further examples of regular expressions with regard to several more features, but the queries we phrased for retrieving cases of 'It' clefting should have given you some impression of the power of regex as a query language.

As with R, and every other programming language for that matter, a word of caution is due. Since because it allows for such precise querying, regex is extremely susceptible to small mistakes and coding differences. For instance, if we replace the ' – which we required it in the search for 'Tis – with an ', AntConc will retrieve zero hits because the character we searched for is not in the corpus. Regex is as nitpicky as any other programming language.

Another thing to be aware of is that differences between corpora can to throw a spanner in the works. For example, all of our queries above included a whitespace between 'It' and ''s'. If you wanted to compare the ICE Ireland with the ICE GB, our search queries would give you zero hits from the ICE GB. Why, you ask? It is not because British English never

uses contractions but because the ICE GB does not include a whitespace between 'It' and ''s'. To get results from ICE GB, we would have to search for:

```
It's the (\w+ )
```

This would retrieve some of the desired hits, such as:

9. s2a-021:51_1:A_1 It's It's It's the Davy Crocketts of this world who are needed to parade about on the battlements

Although the ICE corpora are intended for comparison and share standardizing guidelines, when it comes to the nitty-gritty details, standardization is often insufficient, and care needs to be exerted constantly. You will run into issues like these frequently, so we advise you to be aware and not to despair.

The 'after' perfect

The possibly best investigated feature of Irish English is the after perfect (e.g. Filppula, 1999: 99 ff.), which is considered to have developed due to contact with Irish Gaelic. Surface search string operationalizations to find this feature in corpora fortunately are simple:

```
after \w+ing
```

While there are 71 hits which include some true positives, the precision of this query is very low. Many of our hits are false positives, such as:

10. W2A-011 However, after adopting the approach for one year, I decided to supplement the methodology

But we also get some true positives:

11. S1A-046:100:A_ And he's after coming back from England you know

In order to tauten our search, we can also formulate a more precise query which finds frequent forms of the auxiliary 'be' followed by 'after' and an '-ing' form.

```
(m|be|was|is|'s) after \w+ing
```

This returns eight results, of which six are true positives.

Although we so far looked for after perfect constructions by using key words like "after + ing", the after perfect can also occur in noun phrases, such as "We're just after our dinner" or "The baby is just after her sleep". To search for these, we can use a query like:

```
after (\w+){1,5} (dinner|breakfast|tea|lunch|sleep|nap|beer|walk)
```

Which gives us a single hit:

12. S1A-008:112:A_ I'm not not that long after my dinner

If you are working with tagged data, you do not have to write a sequence of alternatives the way we do here. Instead, you could simply search for 'after' followed by an NP tag with several words in between.

And, of course, if you are working with untagged data, you can use a tagger to process your corpus. We do not discuss part-of-speech tagging in detail in this book, but we want to point out that there are tools which allow you to tag your corpora, such as the TreeTagger (https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/).

Singular existential with plural NP

The singular existential (see Hickey, 2005: 121 and Walshe, 2009) is another well-known Irish feature. A simple surface operationalization can be made by the following query:

```
there 's \w+s
```

This retrieves 81 hits, of which about two thirds are false positives. True positives including sentences like:

13. S1A-027:177:C_ I'm sure there's loads of cafes saying that they're the they're the
14. S1A-028:52:C_ But there's lots of uhm like I mean say if you were going to analyse a a rock face I mean there's probably only one way you can actually analyse it

The true positives are dominated by sentences containing 'loads' and 'lots', which can be seen as lexicalized predeterminers, but also abstract nouns and even some animate nouns, as in the last example, can be found.

If we substitute the contraction in our query by 'is', we get another 40 hits of which 4 are true positives, including:

15. S1A-038:B_ But there is gates and I would assume

Again, this gives us a sense for how nitpicky regex is, but also demonstrates its usefulness.

Exercise

[H5P Exercise: Chi-square test, semantic classification and more search patterns: However, if we enter the same search query in ICE GB, we get 31 hits, including sentences like "And there's examples of the damage to those which required the building to be closed with the possibility of demolition involved in that case" (s2a-025:91_1:A_1). This leaves us in one of these situations where one group seems to use a feature substantially more than the other, but it's not quite clear whether the difference is significant. Accordingly, we should perform a significance test.]

Outlook/Afterword

...

References:

Anthony, Laurence. 2019. *AntConc* (Version 3.5.8) [Computer Software]. Tokyo, Japan: Waseda University. Available from https://www.laurenceanthony.net/software

Filppula, Markku. 1999. *The grammar of Irish English. Language in Hibernian style*. London, Routledge.

Heckendorn, Robert. Regular expression primer.

Hickey, Raymond. 2005. *Dublin English: Evolution and Change. Varieties of English around the world*. Amsterdam/ Philadelphia: Benjamins.

Greenbaum, Sidney. (ed.) 1996. Comparing English Worldwide: The International Corpus of English. Oxford: Oxford University Press.

Schneider, Gerold. 2013. Describing Irish English with the ICE Ireland Corpus. University of Lausanne. Institut de Linguistique et des Sciences du Langage. *Cahiers*, 38: 137–162.

TreeTagger.

Trudgill, Peter and Jean Hannah. 2002. *International English: A Guide to the Varieties of Standard English* (4th ed). London, Arnold.

Walshe, Shane. (2009). *Irish English as Represented in Film*. Frankfurt, Peter Lang.